NATIONAL TECHNICAL UNIVERSITY OF ATHENS
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION OF COMPUTER SCIENCE

Multidimensional Semistructured Data

Representing and Querying Context-Dependent Multifaceted Information on the Web

Yannis Stavrakas

This dissertation is submitted at National Technical University of Athens as part requirement for the degree of Doctor of Philosophy in Computer Science.

Athens, Greece, June 2003

Η διατριβή διατίθεται σε αγγλική (πρωτότυπη) και ελληνική έκδοση.
The dissertation is available in English (original) and Greek editions.

NATIONAL TECHNICAL UNIVERSITY OF ATHENS

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION OF COMPUTER SCIENCE

Multidimensional Semistructured Data

Representing and Querying Context-Dependent Multifaceted Information on the Web

Dissertation submitted as part requirement for the degree of
Doctor of Philosophy in Computer Science
by

Ioannis (Yannis) S. Stavrakas

B.Sc. in Physics, University of Athens, 1989
M.Sc. with Distinction in Computer Science, University College London, 1992

Supervisor: Professor Timos Sellis

Approved by the thesis examination committee on the 11th of June 2003.

Timoleon Sellis Professor NTUA	Ioannis Vassiliou Professor NTUA	Andreas Stafilopatis Professor NTUA
Panagiotis Tsanakas Professor NTUA	Foto Afrati Professor NTUA	Panos Constantopoulos Professor Univ. of Crete

Manolis Gergatsoulis
Assis. Prof. Ionian Univ.

IOANNIS S. STAVRAKAS

... to my parents

Από την Φαντασίαν έως εις το Χαρτί. Είναι δύσκολον πέρασμα, είναι επικίνδυνος θάλασσα. Η απόστασις φαίνεται μικρά κατά πρώτην όψιν, και εν τοσούτω πόσον μακρόν ταξίδι είναι, και πόσον επιζήμιον ενίοτε δια τα πλοία τα οποία το επιχειρούν.

Η πρώτη ζημία προέρχεται εκ της λίαν ευθραύστου φύσεως των εμπορευμάτων τα οποία μεταφέρουν τα πλοία. Εις τας αγοράς της Φαντασίας, τα πλείστα και τα καλύτερα πράγματα είναι κατασκευασμένα από λεπτάς υάλους και κεράμους διαφανείς, και με όλην την προσοχήν του κόσμου πολλά σπάνουν εις τον δρόμον, και πολλά σπάνουν όταν τα αποβιβάζουν εις την ξηράν. Πάσα δε τοιαύτη ζημία είναι ανεπανόρθωτος, διότι είναι έξω λόγου να γυρίσει οπίσω το πλοίον και να παραλάβει πράγματα ομοιόμορφα. Δεν υπάρχει πιθανότης να ευρεθεί το ίδιον κατάστημα το οποίον τα επώλει. Αι αγοραί της Φαντασίας έχουν καταστήματα μεγάλα και πολυτελή, αλλ' όχι μακροχρονίου διαρκείας. Αι συναλλαγαί των είναι βραχείαι, εκποιούν τα εμπορεύματά των ταχέως, και διαλύουν αμέσως. Είναι πολύ σπάνιον έν πλοίον επανερχόμενον να εύρη τους αυτούς εξαγωγείς με τα αυτά είδη.

Μία άλλη ζημία προέρχεται εκ της χωρητικότητος των πλοίων. Αναχωρούν από τους λιμένας των ευμαρών ηπείρων καταφορτωμένα, και έπειτα όταν ευρεθούν εις την ανοικτήν θάλασσαν αναγκάζονται να ρίψουν έν μέρος εκ του φορτίου δια να σώσουν το όλον. Ούτως ώστε ουδέν σχεδόν πλοίον κατορθώνει να φέρη ακεραίους τους θησαυρούς όσους παρέλαβε. Τα απορριπτόμενα είναι βεβαίως τα ολιγοτέρας αξίας είδη, αλλά κάποτε συμβαίνει οι ναύται, εν τη μεγάλη των βία, να κάμνουν λάθη και να ρίπτουν εις την θάλασσαν πολύτιμα αντικείμενα. (...)

Θλιβερόν, θλιβερόν είναι άλλο πράγμα. Είναι όταν περνούν κάτι πελώρια πλοία, με κοράλλινα κοσμήματα και ιστούς εξ εβένου, με αναπεπταμένας μεγάλας σημαίας λευκάς και ερυθράς, γεμάτα με θησαυρούς, τα οποία ούτε πλησιάζουν καν εις τον λιμένα είτε διότι όλα τα είδη τα οποία φέρουν είναι απηγορευμένα, είτε διότι δεν έχει ο λιμήν αρκετόν βάθος δια να τα δεχθή. Και εξακολουθούν τον δρόμον των. Ούριος άνεμος πνέει επί των μεταξωτών των ιστίων, ο ήλιος υαλίζει την δόξαν της χρυσής των πρώρας, και απομακρύνονται ηρέμως και μεγαλοπρεπώς, απομακρύνονται δια παντός από ημάς και από τον στενόχωρον λιμένα μας. (...)

Κ.Π. Καβάφης, «Τα πλοία».

Acknowledgements

A number of people have helped me in various ways to complete this work. First of all, I wholeheartedly thank my supervisor Timos Sellis, professor at National Technical University of Athens and director of the Knowledge and Database Systems Laboratory (DBLab). For his support, guidance, and encouragement. For setting high scientific standards. For the time and effort he spent on my work, especially at the later stages. Most important, for a mixture of balance, professionalism, and integrity he emits, imposing an environment one can only feel privileged to be part of. My thanks also go to Yannis Vassiliou and Andreas Stafilopatis, professors at National Technical University of Athens, for serving in my thesis comittee.

Another person I want to thank is Manolis Gergatsoulis, assistant professor at the Ionian University. For his collaboration in formulating a number of answers related to this work, and an even greater number of questions, especially in the first difficult steps. For the fun we had burning the midnight oil while preparing papers for submission. His help in establishing the direction towards the present work has been indispensable. In addition, I thank Panos Constantopoulos, professor at the University of Crete, Panagiotis Tsanakas, professor at NTUA, and Foto Afrati, professor at NTUA, for serving in my thesis examination committee.

I also want to thank Costas Spyropoulos, research director of the Software and Knowledge Engineering Laboratory (SKEL) at the National Center for Scientific Research "Demokritos". For tolerating my disrupted pace when jumping from project development to research and vice-versa. But mostly, for believing in me during these years. I would also like to thank Panos Rondogiannis, assistant professor at the University of Athens, for opening a discussion that was to lead me to the topic of the present work.

Next, I would like to thank the students at National Technical University of Athens whose diploma thesis I supervised: Kostis Pristouris, Antonis Efandis, Christos Doulkeridis, Vassilis Zafeiris, Costas Karageorgos, Costas Valakas, Andreas Polyrakis, Vangelis Zorbas, Costas Chatzinas, and Cristian Chereji. The first four of them, together with Panos Georgantas and Stefanos Souldatos, contributed to the MSSD implementations described in this work. Moreover, Dimitris Karteris, Athina Mouzaki, Theodoros Mitakos, and Dimitris Sterpis influenced the shape of MXML, and our collaboration in the research project ΠΕΝΕΔ-99ΕΔ265 at N.C.S.R "Demokritos" resulted in the implementation of "MXML Web Server", described in this work. I thank all of them. The members of DBLab have helped me, through various discussions, to formulate a background on which to base the quest for a research topic. I thank them all, especially those closer to my own research interests: Theodore Dalamangas, Vangelis Zorbas, Dinos Arkoumanis, Spiros Skiadopoulos, Yorgos Adamopoulos, Panos Xeros, to name just a few.

The Institute of Communication and Computer Systems (ICCS) supported me with a grant in the first year of my research. I am grateful to them.

Moreover, I want to thank my mother Chrysavgi, my father Spiros, and my brother Panos for their support, and for bearing my absence without complain. Finally, my cordial thanks to Efi Gazi for her continuous encouragement and support.

Table of Contents

List of Figures

ΣΥΝΟΨΗ

Ενώ στις παραδοσιακές βάσεις δεδομένων και στα αντίστοιχα πληροφοριακά συστήματα το πλήθος των χρηστών είναι λίγο πολύ γνωστό και το υπόβαθρό τους είναι σε μεγάλο βαθμό ομογενές, οι χρήστες του Παγκόσμιου Ιστού δεν έχουν κοινό υπόβαθρο ούτε όμοιες θεωρήσεις όταν επεξεργάζονται και ερμηνεύουν δεδομένα. Αυτοί οι χρήστες μπορεί να έχουν διαφορετικές οπτικές των ίδιων πληροφοριακών οντοτήτων, κάτι που θα πρέπει να ληφθεί υπόψη από τα μοντέλα δεδομένων και τις γλώσσες επερωτήσεων για τον Παγκόσμιο Ιστό. Επιπλέον, οι παροχείς πληροφορίας χρειάζεται συχνά να διαχειριστούν παραλλαγές των ίδιων ουσιαστικά δεδομένων, οι οποίες απευθύνονται σε διαφορετικές ομάδες καταναλωτών.

Σε αυτήν την διατριβή υποστηρίζουμε ότι τα δεδομένα του Ιστού θα πρέπει να μπορούν να προσαρμόζονται σε διαφορετικά *ερμηνευτικά περιβάλλοντα* [contexts], και ότι η δυνατότητα αυτή θα πρέπει να διατίθεται με τρόπο ευέλικτο και εννιαίο στο επίπεδο των συστημάτων βάσεων δεδομένων. Προτείνουμε τα *πολυδιάστατα ημιδομημένα δεδομένα* (*MSSD*) [multidimensional semistructured data], όπου πληροφοριακές οντότητες μπορούν να εκδηλώνουν διαφορετικές εκφάνσεις, ανάλογα με το ερμηνευτικό περιβάλλον. Καταρχήν, ορίζουμε έναν απλό και ευέλικτο τρόπο για να εκφράζουμε ερμηνευτικά περιβάλλοντα χρησιμοποιώντας *διαστάσεις* [dimensions]. Σύμφωνα με την προσέγγισή μας, το ερμηνευτικό περιβάλλον αντιπροσωπεύει μιά σειρά από εναλλακτικούς *κόσμους* [worlds], σε κάθε έναν από τους οποίους οι πληροφορίες αποκτούν μια συγκεκριμένη ερμηνεία. Ορίζουμε πράξεις ανάμεσα από ερμηνευτικά περιβάλλοντα, και εξετάζουμε τις ιδιότητές τους σε βάθος. Στην συνέχεια, προτείνουμε ένα μοντέλο δεδομένων για MSSD που ονομάζεται *Πολυδιάστατος Γράφος Δεδομένων* [Multidimensional Data Graph], και ερευνούμε διεξοδικά τις ιδιότητές του. Ο Πολυδιάστατος Γράφος Δεδομένων επεκτείνει το Object Exchange Model (OEM), ένα μοντέλο γράφου για ημιδομημένα δεδομένα, προκειμένου να υποστηρίξει με άμεσο τρόπο πληροφορία που παρουσιάζει πολλαπλές εκφάνσεις. Το *Πολυδιάστατο OEM* (*MOEM*) [Multidimensional OEM] είναι μια ειδική περίπτωση Πολυδιάστατου Γράφου Δεδομένων, και μπορεί να θεωρηθεί σαν μια συνένωση από συμβατικούς OEM γράφους που υφίστανται κάτω από διαφορετικούς κόσμους. Οι συγκεκριμένοι OEM γράφοι είναι δυνατόν να ανακτηθούν μέσω μιας διαδικασίας που λέγεται *αναγωγή* [reduction], και η οποία μετασχηματίζει τον MOEM στον συμβατικό OEM γράφο που υφίσταται κάτω από έναν δεδομένο κόσμο. Κατόπιν, ορίζουμε την *Πολυδιάστατη Γλώσσα Επερωτήσεων* (*MQL*) [Multidimensional Query Language], μια γλώσσα επερωτήσεων για Πολυδιάστατους Γράφους Δεδομένων και MOEM γράφους που έχει σαν κεντρική έννοια το ερμηνευτικό περιβάλλον, και μπορεί να εκφράσει με κομψό τρόπο *επερωτήσεις οδηγούμενες από περιβάλλοντα* [context-driven queries]. Το μοντέλο MOEM μαζί με την γλώσσα επερωτήσεων MQL επιτρέπουν τη διατύπωση *δια-κοσμικών επερωτήσεων* [cross-world queries], οι οποίες δεν μπορούν να εκφραστούν με συμβατικά μοντέλα δεδομένων και συμβατικές γλώσσες επερωτήσεων. Η MQL έχει υλοποιηθεί χρησιμοποιώντας σαν βάση το σύστημα LORE, και αυτή η υλοποίηση δίνει την ευκαιρία για σύγκριση με «ισοδύναμες» επερωτήσεις στην γλώσσα Lorel.

Αξιολογήσαμε το παραπάνω πλαίσιο που αναπτύξαμε για τα MSSD εφαρμόζοντάς το σε ένα κλασσικό πρόβλημα των βάσεων δεδομένων: χρησιμοποιήσαμε το μοντέλο MOEM για

να αναπαραστήσουμε το ιστορικό βάσεων δεδομένων OEM, και στην συνέχεια την MQL για να θέσουμε επερωτήσεις στο ιστορικό αυτό. Το MOEM και η MQL αποδείχθηκαν ικανά όχι μόνο για να αναπαραστήσουν και να επερωτήσουν το ιστορικό βάσεων OEM, αλλά και το ιστορικό βάσεων MOEM. Επιπλέον, σαν παράδειγμα του πώς οι έννοιες που εισαγάγαμε μπορούν να επεκταθούν σε άλλες κατευθύνσεις, παραθέτουμε την *Πολυδιάστατη XML* (*MXML*) [Multidimensional XML], μια εκδοχή της XML που ενσωματώνει ερμηνευτικά περιβάλλοντα. Για την παροχή και παρουσίαση πληροφορίας που ενέχει πολλαπλές εκφάνσεις προτείνουμε το *πολυδιάστατο παράδειγμα* [multidimensional paradigm], και επιδεικνύουμε μια σχετική πρότυπη υλοποίηση. Τέλος, παρουσιάζουμε ένα υλοποιημένο σύστημα για τη διαχείριση MSSD. Το σύστημα αυτό εμπεριέχει το υποσύστημα *MSSDesigner*, ένα γραφικό περιβάλλον για την εύκολη δημιουργία Πολυδιάστατων Γράφων Δεδομένων, καθώς και για την εκτέλεση επερωτήσεων διατυπωμένων στην MQL.

Εκτιμούμε ότι τα MSSD αντιμετωπίζουν ένα σημαντικό πρόβλημα των βάσεων δεδομένων στο σημερινό παγκοσμιοποιημένο περιβάλλον, και ότι έχουν τη δυνατότητα να εφαρμοστούν πρακτικά σε διάφορες περιοχές. Επιπλέον, τα MSSD ανοίγουν μια σειρά από ενδιαφέρουσες κατευθύνσεις για περαιτέρω έρευνα.

ABSTRACT

Contrary to traditional databases and information systems where the number of users is more or less known and their background is to a great extent homogeneous, Web users do not share the same background and do not apply the same conventions when interpreting data. Such users can have different perspectives of the same information entities, a situation that should be taken into account by Web data models and query languages. Moreover, information providers often need to manage variations of essentially the same data, which are targeted to different consumer groups.

In this thesis we argue that Web data should be able to adapt to different *contexts*, and that this capability should be managed in a flexible and uniform way at the level of database systems. We propose *multidimensional semistructure data* (*MSSD*), where information entities may manifest different facets, according to context. First of all, an intuitive and flexible way to express context through *dimensions* is defined. Context is viewed as representing alternative *worlds*, in which information obtains an unambiguous interpretation. Operations between contexts are introduced, and their properties are examined in detail. Then, a data model for MSSD called *Multidimensional Data Graph* is proposed, and its properties are investigated exhaustively. Multidimensional Data Graph extends the Object Exchange Model (OEM), a graph model for semistructured data, in order to explicitly support multifaceted information. *Multidimensional OEM* (*MOEM*) is a special case of Multidimensional Data Graph, and can be seen as a conglomeration of conventional OEMs holding under different worlds. Those OEMs can be obtained through the process of *reduction*, which transforms a Multidimensional OEM to a conventional OEM holding under a given world. Next, we define *Multidimensional Query Language* (*MQL*), a query language for Multidimensional Data Graphs and MOEMs that treats context as first-class citizen, and can express *context-driven queries* in an elegant way. MOEM and MQL allow the formulation of *cross-world queries*, which have no counterpart in conventional data models and query languages. MQL has been implemented on top of LORE, and this implementation provides a comparison with "equivalent" Lorel queries.

To assess our approach, we applied the MSSD framework we have developed to a classical problem in databases: we used MOEM and MQL to represent and query histories of OEM databases. MOEM and MQL proved powerful enough to represent and query not only OEM histories, but MOEM histories as well. Moreover, as an example of how the concepts we introduce can span other directions, we discuss *Multidimensional XML* (*MXML*), a version of XML that incorporates context. For manipulating and viewing multifaceted information we propose the *multidimensional paradigm*, and demonstrate a prototype implementation. Finally, an implemented system for managing MSSD is presented. This system encompasses *MSSDesigner*, a graphical user interface that supports the interactive creation, experimentation, and querying of Multidimensional Data Graphs through MQL.

We believe that MSSD address an important problem of databases in today's global environment, and that they have the potential to be practically applied in various domains. Moreover, MSSD open a number of interesting directions for further research.

1 INTRODUCTION

Semistructured data (SSD in short) is a term coined by the database research community to describe information that has irregular structure. Most of the information available today falls into this category, including HTML and XML Web data. A number of models and query languages have been proposed over the recent years for representing and querying SSD. In this thesis, we claim that the notion of *context* is pivotal for databases in a global environment such as the Web, and we propose a new data model and query language for SSD where context plays a central role.

In the present chapter, we state the problem and motivate our approach. Then, we list the main contributions of this thesis, and we refer to material that has been published up to the moment of writing these lines. Finally, we close the chapter with a thesis outline.

1.1 THE PROBLEM, AND THESIS CONTRIBUTION

The nature of the Web posed a number of new problems [BBC+98] to the database research community. One such problem is that, while in traditional databases and information systems the number of users is more or less known and their background is to a great extent homogeneous, Web users do not share the same background and do not apply the same conventions when interpreting data. Such users can have different perspectives of the same entities, a situation that should be taken into account by Web data models and query languages. A related issue is that information providers often need to manage different variations of essentially the same data, which are targeted to different consumer groups. Similar problems also appear when integrating information from various sources [GPQ+97], where the same conceptual entity may exhibit different structure, or contain conflicting data.

Those problems call for a way to represent and handle information entities that manifest different facets, whose contents can vary in structure and value. As a simple example imagine a product (car, laptop computer, etc.) whose specification change according to the country it is being exported. Or a Web page that is to be displayed on devices with different capabilities, like mobile phones, PDAs, and personal computers. Another example is a report that must be represented at various degrees of detail and in various languages.

A solution would be to create a different report for every possible combination of variations. Such an approach is certainly not practical, since it involves excessive duplication of information. What is more, different variants are not associated as being parts of the same entity. This prevents the formulation of interesting queries that involve facets of information entities, as for example "*give me in low detail the reports whose high detail facet satisfies the specified condition*".

As a motivating example consider an on-line bookstore that has a variety of clients, which have different needs and are treated differently according to the bookstore policy. Students, for instance, need to browse books by title or author, and are charged lower prices than

libraries. For libraries, on the other hand, the ISBN of a book is enough. The bookstore would like to present information that is tailored to the client that placed the request:

Book ISBN: 0-13-110362-8, Publisher: Prentice Hall, Price: 29.4
Book Title: The C programming language, Authors: Brian W. Kernighan, Dennis M. Ritchie, Publisher: Prentice Hall, Price: 13

Both entries above refer to the same book. The first corresponds to a library request, while the second to a request made by a student. Note that the customer type (library or student) affects not only the value of some objects (e.g. "Price"), but the structure of data as well: a different portion of the available data is presented, according to the case at hand. In addition to customer type, another factor is associated with the way information is *displayed*, and the second entry above uses letters of greater size for the word "Book"[1].

Such behavior is not uncommon for Web applications. However, the capability to adapt information to characteristics of the prospective consumer is currently part of the application logic, and must be programmed from scratch in every different case. What we envisage is that *the capability to adapt becomes part of the information itself, managed in a uniform and flexible way by database systems*. This will alleviate the burden of developing ad-hoc solutions; instead, applications will pose simple queries, which will be aware of the dependency of information facets from parameters like "customer type".

Although existing Web data models such as OEM [PGW95, AQM+97, Suc98], and query languages such as Lorel [GPQ+97, AQM+97] are in principle capable of representing and querying multifaceted entities, they fall short for the reasons that follow[2].

(a) Hidden semantics: by not addressing directly the issue of multiple facets, it is the responsibility of an application to assign semantics in an ad-hoc manner.

(b) Cumbersome notation: representing multiple facets of an entity cannot be done in an elegant way.

(c) Duplication of information: an ad-hoc approach may lead to duplicating information that is common, which is undesirable.

The need for multifaceted entities exceeds the boundaries of Web data. It can be seen in a wide spectrum of diverse situations, which occasionally adopt ad-hoc solutions of limited effectiveness. A tangible example of such a case that shows the applicability of multifaceted information is encountered in *Microsoft PowerPoint 2000*, a computer program for creating and displaying presentations, which includes a feature called *custom shows*. This feature allows to "...create a presentation within a presentation. Instead of creating multiple, nearly identical presentations for different audiences, you can group together and name the slides

[1] In Chapter 7 we discuss this example in detail, and explain the mechanism used to generate and display variants of an information entity.
[2] These shortcomings become evident in Chapter 5, where we investigate how OEM and Lorel can be used to represent and query multifaceted information.

that differ and then jump to these slides during your presentation"[3]. *Custom shows* do not allow to include or exclude *portions of a single slide* according to the intended audience, which leads to duplicating across many slides the portions that target more that one audience. Ideally, we would like to be able to mark any part(s) and component(s) of a presentation as targeted to particular audience(s), and subsequently specify an audience to have the presentation automatically tailored to that audience. Moreover, it would be nice to be able to pose useful questions, like "*show me those components of slides no. 3 to 8 which are targeted to audiences A and B, but not to audience C*". To go even further, except the intended audience, other factors may affect a presentation as well; for example, consider the case where the presentation duration should adhere to different time limits depending on the situation. In this case, pieces of the presentation material could be assigned different degrees of priority or importance, allowing low priority information to be automatically omitted when the time limit is short.

Information that presents different facets is also encountered in *probabilistic databases*, where facets are associated with some measure of uncertainty as to whether they express the real world accurately or not. A relatively recent probabilistic extension of the relational model can be found in [DS96], while in [DS98] the corresponding extension to SQL is presented. Probabilistic features have lately been added to XML [NJ02], however the uncertainty measure that qualifies facets has a specialized semantics that is not general enough to cover other factors that may characterize variants of data in the Web.

Our work was influenced by Intensional HTML [WBSY98, Bro98, Yil97, Bro98a], or IHTML in short, a Web authoring language that allows a single Web page to have different variants and to dynamically adapt itself to a given **world** (specified by assigning single values to a number of variables called **dimensions**). IHTML follows a document-centric approach, using a naming scheme to encode worlds in the filenames of document versions. Our work follows a different direction: (a) our central concept is **context**, which represents a *set* of possible worlds, while dimensions used to express a context can be assigned many values and be combined in conjunctions and disjunctions, (b) we formalize the notion of context and introduce operations between contexts, and (c) we focus on a data-centric approach, which aims to incorporate context in data models and query languages for semistructured data.

In this thesis we propose **multidimensional semistructured data** (**MSSD** in short), which are semistructured data that present different facets under different **contexts**. We express context in a flexible and intuitive way, by assigning values to variables called **dimensions**. The main difference between conventional and multidimensional semistructured data is the introduction of **context specifiers**. Context specifiers are syntactic constructs that are used to qualify pieces of semistructured data and specify sets of **worlds** under which those pieces hold. A world is an environment under which data obtain an unambiguous interpretation, and is defined by assigning a single value to every dimension.

In this way, it is possible to have at the same time variants of the same information entity, each holding under a different set of worlds. An information entity that encompasses a number of variants is called **multidimensional entity**, and its variants are called **facets** of the entity. The facets of a multidimensional entity may differ in value and / or structure, and can in turn be multidimensional entities or conventional information entities. Each facet is associated with a context that defines the conditions under which the facet becomes a **holding facet** of the multidimensional entity.

The main contributions of this thesis are:

[3] Extracted from the on-line help section of Microsoft PowerPoint 2000 entitled "About custom shows".

1. An intuitive and flexible way to express context through **dimensions** is defined. Context is viewed as representing alternative **worlds**, in which information obtains an unambiguous interpretation. Operations between contexts are introduced, and their properties are examined in detail.
2. **Multidimensional Data Graph**, a data model for MSSD is proposed, and its properties are investigated exhaustively. **Multidimensional OEM** (**MOEM**) is a special case of Multidimensional Data Graph, and can be seen as a conglomeration of conventional OEMs holding under different worlds. Those OEMs can be obtained through the process of **reduction to OEM**, a flexible mechanism for information adaptation, which transforms a Multidimensional OEM to a conventional OEM holding under a given world.
3. **Multidimensional Query Language** (**MQL**) is defined. MQL is a query language for Multidimensional Data Graph and MOEM that treats context as first-class citizen, and can express *context-driven queries* in an elegant way. MOEM and MQL allow the formulation of *cross-world queries*, which have no counterpart in conventional data models and query languages. MQL has been implemented on top of LORE, and this implementation provides a comparison with "equivalent" Lorel queries.
4. To assess our approach, we apply the MSSD framework we have developed to a classical problem in databases: we use MOEM and MQL to represent and query histories of OEM databases. MOEM and MQL proved powerful enough to represent and query not only OEM histories, but MOEM histories as well.
5. As an example of how the concepts we introduce can span other directions, we discuss **Multidimensional XML** (**MXML**), a version of XML that incorporates context. For manipulating and viewing multifaceted information we propose the **multidimensional paradigm**, and demonstrate a prototype implementation.
6. An implemented system for managing MSSD is presented. This system encompasses *MSSDesigner*, a graphical user interface that supports the interactive creation, experimentation, and querying of Multidimensional Data Graphs through MQL.

The role of context in managing the multiple aspects of data in today's global environment is attracting increasing attention, especially in the frame of *ubiquitous computing* and *mobile computing* [CK00]. We believe that MSSD address an important problem of databases in the Web, and open interesting directions for further research. Moreover, MSSD have the potential to be practically applied in various domains. For example, the web-publishing platform *OMSwe* [NP03, NP03a] is based on an Object DBMS, which has been extended to support a flexible, domain-independent model for information delivery where context plays a pivotal role. Thus, an interesting direction is to use MSSD concepts in problems from disperse fields, like: in providing personalized information and services, by adapting them to the profiles of consumers; in information integration, for modeling objects whose value or structure vary according to sources; in digital libraries, for representing metadata that conform to similar formats; in representing geographical information, where possible dimensions could be *scale* and *theme*.

Part of the work presented in this thesis has been published or is submitted for publication in [SG02, SPES03, SGDZ02, SGDZ03, GSK01, GSK+01, MGSI01a, MGSI01, GS03, SGM00, SGR00, ZDSG02, EPS03].

1.2 THESIS OUTLINE

This thesis is structured as follows. In Chapter 2 we review some preliminary work on semistructured data and XML, focusing on issues that are mentioned in subsequent chapters. We state the definition of OEM, and give examples of Lorel and UnQL queries. We refer briefly to XML, and to related technologies like DTD, XSL, and query languages for XML.

In Chapter 3 we develop a formalism for representing context using variables called **dimensions**. We define **context specifiers** as constraints on the possible values dimensions can take. We give semantics to context by showing that it can be viewed as a set of possible **worlds**, where a world is an environment under which information obtains an unambiguous interpretation. We define a number of context operations, and examine their properties.

In Chapter 4 we introduce **Multidimensional Data Graph**, a graph data model that incorporates context specifiers and treats multifaceted entities as first-class citizens. Multifaceted entities may manifest different facets under different worlds, and context specifiers are used to qualify those facets and define constraints on the worlds under which a facet may hold. We investigate the properties of Multidimensional Data Graph, and define the process of **reduction to OEM**, which extracts from a Multidimensional Data Graph a conventional OEM holding under a given world. Based on the properties of Multidimensional Data Graph, we define **Multidimensional OEM** (**MOEM**) as a special case of Multidimensional Data Graph that is a strict conglomeration of conventional OEMs holding under various worlds.

In Chapter 5 we present **Multidimensional Query Language** (**MQL**), a query language that supports context and multifaceted entities as first class citizens, and can express *context-driven queries*. **Context path expressions** extend path expressions by incorporating **context qualifiers**, which are used to pose context conditions on the database graph. In addition to context path expressions, MQL uses two special clauses for handling context: the clause `within` can express complex conditions on contexts, and the clause `context` creates new context variables to be used in the construction of results. Our prototype implementation of MQL demonstrates how an MQL query compares to an equivalent Lorel query, and shows the benefits of treating context and multifaceted entities as first class citizens.

In Chapter 6 we assess our approach, by applying the MSSD framework we have developed to a classical problem in databases: we use MOEM and MQL to represent and query histories of OEM databases. We introduce the **basic change operations** of MOEM, and specify a process that encodes in an MOEM the history of an OEM database. Moreover, we show that temporal OEM snapshots can be obtained from MOEM using the process of reduction to OEM, and we present *OEM History*, a relevant implemented system. We give a number of MQL query examples on the history of a sample OEM database, and through them we confirm the expressiveness of MQL. An interesting point is that MOEM and MQL can represent and query not only OEM histories, but MOEM histories as well.

Chapter 7 serves as an example of how the concepts we introduce in previous chapters can span other directions, and proposes **Multidimensional XML** (**MXML**), an extension of XML suitable for representing data that assume different facets under different contexts. Moreover, we propose an extension of DTD called **Multidimensional DTD** (**MDTD**) that describes the logical structure of MXML documents. We present a new paradigm called **multidimensional paradigm** for manipulating and displaying multifaceted, context-dependent Web data using **multidimensional XSL** (**MXSL**) stylesheets. We describe a system that implements the basic functionality of the multidimensional paradigm, and demonstrates how a user can interact with a multidimensional document and view different variants under different worlds.

In Chapter 8 we present a system for managing MSSD that implements most of the essential concepts introduced in this thesis, and can be used for the interactive creation, experimentation, and querying of Multidimensional Data Graphs through MQL.

Finally, Chapter 9 concludes the thesis by summarizing our contribution, and identifies directions for future research and possible applications of MSSD.

2 PRELIMINARIES

The impact of the World Wide Web over the last decade cannot be underestimated. Shortly after its invention, it has become a widely accepted means of information exchange, communication, and commerce at a global scale. The Web continues to evolve, with promising technologies like web services [Pap02], and motivating visions for a Semantic Web [SWEB] in which Web data will be not only machine-readable, but machine-understandable as well.

From the start, database technology played an important role in the Web. A substantial part of the Web content is stored in relational databases, which are used to dynamically construct HTML [HTML] pages and to support electronic commerce applications. Because of the need for database support, a number of ways were soon devised to connect databases to the Web [Mal98, SKV98, ZSC99]. But databases offer more than just being a back-end technology. The database research community rapidly recognized [SSU95, SZ96, SZ97] that the Web, as a huge source of information that needs to be accessed and manipulated, could benefit from the experience accumulated in databases during the past decades. Therefore, data models, query languages, query optimization, and concurrency techniques should be adapted to a new framework, posed by the Web. As stated in the Asilomar Report on Database Research [BBC+98], the recommended long-term goal for database research is:

> "The Information Utility: Make it easy for everyone to store, organize, access, and analyze the majority of human information online."

Consequently, databases complement the *document-centric* origins of the Web with a *data-centric* perspective. In what follows, we refer briefly to ways database technology has been applied [FLM98] to the Web. We then discuss semistructured data, which comprise data models and query languages for the Web proposed by the database community. Finally, we refer to XML and related technologies, which, although of document-centric origins, have been adopted by the database community and currently constitute a very active field of database research.

2.1 DATABASES AND THE WORLD WIDE WEB

The success of the Web can be attributed to the fact that it offered a simple and standard way for universal exchange of information. In brief, the key elements of the Web are: (a) HTML [HTML], a markup language that structures text according to its visual rendering and embodies intra-document and inter-document links, (b) URL [URL], a way to identify resources on the internet, and (c) HTTP [HTTP], a simple protocol that allows a client to connect to a server and request a resource.

In this simplified view of the Web, the units of information are HTML documents identified by URLs. In this paradigm, information is assumed to have no structure, and Web search engines employ techniques from Information Retrieval [GRGK97] to obtain documents of interest. Although search engines make use of database technologies [BMPW98, YL96] as

well, they typically follow the Information Retrieval methodology, and view a query as a set of filters based on keywords. This approach gives flexibility and ease of use, however it lacks the accuracy and completeness offered by databases: search engines often return irrelevant documents, while they often not recognize relevant documents as such.

On the other hand, the *database culture* [ABS00] suggests the use of data models, such as entity-relationship diagrams and relational database schemas, to capture the structure of information, together with the use of query languages that rely on data models in order to access information. A principal characteristic of the database perspective is the separation between the logical view, related to models and queries, and the physical view, related to storage and efficiency. This important distinction to levels of abstraction is not recognized by the document-centric view of the Web. Therefore, it makes sense for the data-centric and the document-centric view of the Web to converge and benefit from each other.

A number of information integration systems have been developed, in an effort to give an expressive and homogeneous query interface to heterogeneous information sources. Examples of such systems are WHIRL [Coh98, Coh98a, Coh98b, CH98], Information Manifold [KLSS95, LMSS95, LSK95], TSIMMIS [PGW95, GPQ+97, CGH+94, GHI+95, PGU96], and Garlic [CHS+95, HKWY96, HMN+97, RS97]. Databases attempted to adapt to the Web by following a multi-tier architecture instead of the classical client-server architecture. In this multi-tier architecture the server is the data repository, the client interprets and displays data, and a *middleware* tier uses mediators [Wie92] for integrating data from various sources and for transforming data to a suitable form. Mediated information integration systems can be classified [Ull97, GP98] according to how a global schema is defined: (a) as a conglomeration of views describing individual sources, or (b) using global predicates and collections to which local sources are mapped.

The TSIMMIS approach uses a network of mediators and wrappers to extract information and to create a unified view of the underlying sources. The data model introduced by TSIMMIS is called Object Exchange Model, or OEM, and the language used for querying is Lorel [GPQ+97, AQM+97]. OEM and Lorel are also used in Lightweight Object Repository [MAG+97], or LORE, a database management system made especially for semistructured data.

Another direction of research is to apply database methodologies for the management of Web sites. A methodology for designing hypermedia applications is presented in [ISB95]. STRUDEL [FFK+98, FFLS97, FFLS98] is a system for creating and managing Web sites, which separates the physical organization of data, the logical view of the site, and presentation issues. OEM is used as its data model, and StruQL is its query language, which specifies in a declarative way the structure of sites. Another system for Web site development is ARANEOUS [AMM97, AMM97a, MMAC99], which proposes the data model ADM for describing hypertext collections, the language ULIXES for defining views, and incorporates a set of tools that undertake the construction of Web sites.

A major contribution of this convergence between databases and the Web is the work on semistructured data and XML. In what follows, we review some of the work on data models and query languages for semistructured data and XML, focusing on topics that will be used in subsequent chapters.

2.2 SEMISTRUCTURED DATA

In traditional databases, information is well structured and conforms to a fixed schema defined in advance. Database management systems use the schema to store and index data, and to process queries and updates. On the other hand, Web data may lack regular structure

and may not conform to a separate schema. Semistructured data [Suc98, Bun97, Suc97], or SSD, is a term coined to describe information that is less structured than relational or object oriented data, but nevertheless is not completely unstructured. Examples of semistructured data are LaTex and BibTex files, RTF, HTML, and XML files.

Semistructured data may exhibit the following particularities.

- *Variations in structure*: Data heterogeneity may take the form of missing fields (the address of a person is missing), duplicated fields (two phone numbers exist for the same person), or changes in representation (some prices may be in Euros while others in US Dollars). Moreover, data can be nested in arbitrary depth, which makes the use of the relational model difficult.
- *Structure may be partial*: Parts of the data may lack structure completely (e.g. images), while other parts may yield little structure (e.g. plain text).
- *Types are only indicative*: Semistructured data do not adopt a strict typing policy like traditional databases.
- *Embedded schema*: In semistructured data the schema is not separate from data like in traditional databases. In other words, the type of objects is not external to object instances. This caused semistructured data to be called *schemaless* and *self-describing* data, in the sense that schema exists as part of the information itself.
- *Schema is large and rapidly evolving*: The more the variations in structure, the larger the schema becomes. In addition, Web data are changing rapidly and unpredictably; therefore schemata are also likely to evolve at a quick rate.
- *A-posteriori dataguide*: In contrast to traditional databases that are populated based on a pre-existing schema, the schema in semistructured data is extracted afterwards to facilitate access to information.

2.2.1 Data Models for SSD

Data models for SSD are based on directed graphs, in order to cope with the irregular structure and deep nesting of semistructured data. A popular model for SSD is Object Exchange Model, or OEM, variations of which have been used in a number of occasions. OEM was first introduced in TSIMMIS [PGW95, GPQ+97] as a graph of interconnected objects. An OEM object is defined in TSIMMIS as comprising an object identifier (oid), a label, a type, and a value. The type can be either *set*, or one of the atomic types (string, integer, etc.). Respectively, the value can be either a set of oids, or an atomic value. Therefore, since nodes in the graph model represent objects, an object of *set* type points through directed edges to the sub-objects contained in its *set* value.

The OEM variation of TSIMMIS attaches labels to nodes (objects). In other OEM variations used in STRUDEL [FFLS97] and LORE [MAG+97, AQM+97] labels are attached to edges rather than nodes. If labels are attached to nodes they become a property of objects; on the other hand, if labels are attached to edges they become a property of the relation between two objects. Attaching labels to edges allows nodes to "see" the same node through different names. An edge-labeled variation of OEM is also used as the data model of UnQL [BDS95, BDHS96, BFS00], a query language for "unstructured" data. In this variation the notion of object equality is based on bisimulation instead of isomorphism.

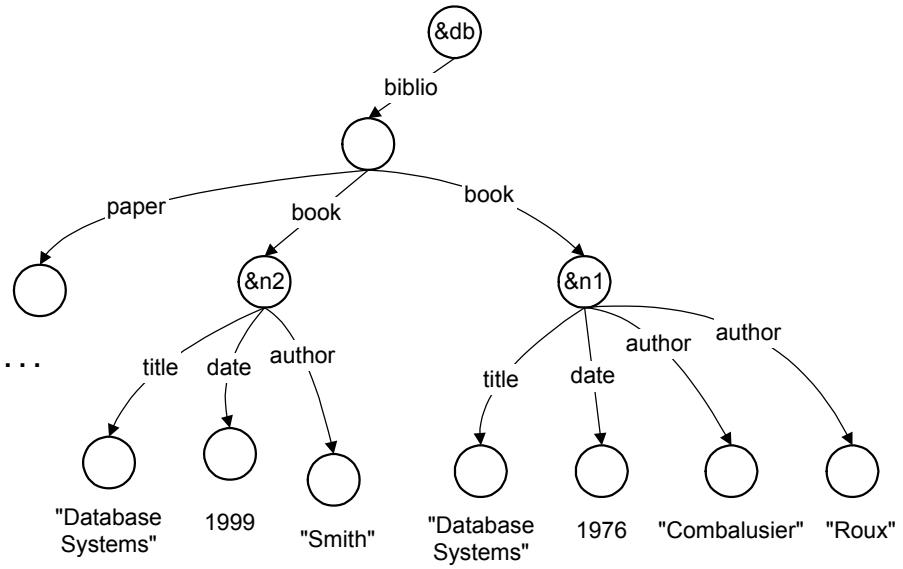
In the frame of this work we will use the edge-labeled variation of OEM. Specifically, we will use the formal definition of OEM given in [Suc98].

Definition 2.1

An OEM graph is a quadruple (V, E, r, v), where the set of nodes V is partitioned into complex and atomic nodes $V = V_c \cup V_a$, the edges are $E \subseteq (V_c \times L \times V)$, $r \in V$ is the root with the property that every node in V is reachable from r, and $v : V_a \rightarrow A$ assigns values to atomic objects; A is the universe of atomic values, and L is the universe of label names.

This definition is equivalent to the slightly different definition of an OEM database presented in [AQM+97]. An OEM example taken from [ABS00] is given in Figure 2.1.

Figure 2.1: A (part of a) bibliographical OEM database.



OEM nodes may have object identifiers (oids); in Figure 2.1, those are `db`, `n1`, and `n2`. By convention, oids in OEM are preceded by the character &. Although the OEM in Figure 2.1 is a tree, in the general case an OEM is a graph. In particular, OEM is a *rooted directed labeled multigraph with valued leaves*.

OEM can be represented textually in various ways [AQM+97, BFS00]. Adopting the format of **ssd-expressions**, appearing in [BFS00, ABS00, Suc98], the OEM of Figure 2.1 is represented as shown in Example 2.1.

Example 2.1

```
&db {biblio: {book: &n1 {author: "Roux",
                         author: "Combalusier",
                         date: 1976,
                         title: "Database Systems"
                        },
              book: &n2 {author: "Smith",
                         date: 1999,
                         title: "Database Systems"
                        },
```

```
            paper:    {...}
          }
    }
```

♦

Curly brackets enclose objects, which may be preceded by an optional object identifier. The syntax of ssd-expressions is formally defined [ABS00] in Table 2.1 in Extended Backus-Naur Form [EBNF], or EBNF for short. Symbols that can be defined by a regular expression start with a capital letter (example: `Oid`), while symbols defined in EBNF start with a lowercase letter (example: `ssd-expr`).

Table 2.1: Syntax of ssd-expression.

```
ssd-expr ::= value | Oid value | Oid
value ::= AtomicValue | "{" complexValue "}"
complexValue ::= LabelName ":" ssd-expr ("," complexValue)?
```

Apart from OEM, a number of other graph models for semistructured data have been proposed. An extension of OEM called Delta OEM, or DOEM, is introduced in [CAW99], addressing the problem of representing histories of changes in OEM databases. In Chapter 6, where we investigate the aforementioned problem in detail, we discuss further [CAW99] and other similar approaches, and we compare them to our own approach. A model for semistructured data that deviates from OEM has been proposed in [BDT98], where edge labels are themselves pieces of semistructured data. In [DBJ99], an extensible semistructured data model has been proposed that uses sets of properties of the form `property_name: property_value` as edge labels. By attaching properties at will, the graph becomes rich in metadata. Different properties may have different semantics, which results in a model of increased generality. The extension to OEM we propose is presented in detail in Chapter 4.

2.2.2 SSD Query Languages

A number of languages have been proposed for querying data on the Web [FLM98, Suc98, Abi97], even before query languages for semistructured data. These can be classified into two categories [FLM98]: *first generation* web query languages, and *second generation* web query languages.

The characteristic of first generation web query languages is that they combine content-based queries with structure-based queries. Content-based search involves the definition of *text patterns* that are matched against the contents of documents, while structure-based search involves the definition of *graph patterns* describing hyperlinks between documents. First generation web query languages include WebSQL [MMM96, MMM97], W3QL [KS95], and WebLog [LSS96]. The following WebSQL example finds all pairs of documents where the first is in the local site and contains the word “database”, and the second is in a remote site and is accessible from the first through a link, whose label is also returned in the results.

```
SELECT d.url, e.url, a.label
FROM Document d SUCH THAT
           “www.mysite.start” -> * d,
           d MENTIONS “database”,
     Document e SUCH THAT
           d => e,
     Anchor   a SUCH THAT
           a.base = d.url
WHERE a.href = e.url
```

WebSQL models the Web as a relational database with two virtual relations: *Document* and *Anchor*. For each document and each anchor a tuple exists in the respective relation. Navigation starts from some known URL, using the symbol `->` to denote navigation in the same server, and the symbol `=>` to denote navigation to some different server.

Second generation web query languages are web data manipulation languages, in the sense that they access the internal structure of web objects and are able to create new complex structures as the results of a query. Their view of web data is very close to the approach taken by semistructured data models. Examples of such languages are StruQL [FFLS97, FFLS98, FFK+98], FLORID [HLLS97], and WebOQL [AM98]. StruQL is the query language of the Web site management system STRUDEL, and uses a clause for binding variables to parts of the input graph database, plus a number of clauses that use Skolem functions to construct a new graph as the result. FLORID is a declarative language based on F-logic, a formalism for object-oriented models. WebOQL focuses on graph restructuring operations, and supports *webs* as a data type, which allow to model sets of related hypertrees.

The most influential languages for semistructured data are UnQL [BFS00, BDHS96, BDS95] and Lorel [AQM+97, GPQ+97, GMW99]. UnQL uses patterns that resemble ssd-expressions to bind variables to nodes in the `where` clause, and also to construct the result graph in the `select` clause:

```
select {paper: {title: X, date: Y}}
where {biblio.paper: {title: X, date: Y}} in db,
      Y > 1995
```

Evaluated on the database of Figure 2.1, the above query returns the title and the publication date of papers that have been published after 1995. Except from normal variables that bind to nodes, UnQL uses *label variables* that bind to edge labels; using label variables makes it possible to transform database metadata into regular data in the results. UnQL queries are translated into *structural recursion*, a construct that reminds of patterns in functional programming languages and allows expressing both queries and transformations in the same formalism.

Lorel focuses on the use of expressive *path expression* constructs and of type *coercions*, in order to facilitate the formulation of queries and make the language more intuitive. The following query returns the title and the author(s) of books or papers that have been published after 1972 and whose title contains “Database”.

```
select X.author, X.title
from db.biblio(.book | .paper) X
where X.date > 1972
      and X.title grep "Database"
```

Evaluated on the database of Figure 2.1, this query returns an answer equivalent to the ssd-expression that follows:

```
{answer: {book:  {author: "Roux",
                  author: "Combalusier",
                  title: "Database Systems"
                 },
          book:  {author: "Smith",
                  title: "Database Systems"
                 },
          paper: {...}
         }
}
```

This example demonstrates a number of Lorel features. First, the path expression in the `from` clause borrows the syntax of regular expressions [Fri97] to match both books and papers in the database. Nodes bound to the variable `X` in the `from` clause are filtered in `where`: path expressions in the `where` clause imply an existential quantifier like "exists a date in `X` which is greater than 1972". Next, a path expression in the `select` clause implies a nested query, which in the case of `X.author` causes all the authors of a single book or paper to appear under a single book or paper object in the results. The label `answer` is the default label of Lorel in results, while labels `book` and `paper` are determined at run-time, based on the matching data path.

In [ABS00] a "core language" for semistructured data is outlined, which combines features from both UnQL and Lorel. Together with Lorel and UnQL, we use this core language as a base for building **Multidimensional Query Language** in Chapter 5.

2.3 XML

Another way to represent semistructured data is eXtensible Markup Language [XML, Wal97, Cha99], or XML. XML is proposed by the World Wide Web Consortium [W3C], or W3C, as a flexible and adaptable means of information exchange over the Web. In contrast to Hypertext Markup Language [HTML], or HTML, where elements and attributes are predefined and deal with how information is to be presented, XML can define new elements and attributes at will, adapting itself to different domains. In addition, XML focuses on the structure of information, leaving presentation issues to be dealt with at subsequent steps. XML solves the problem of parsing data on the Web in an ad-hoc manner, and provides a framework for developing standard tools for extracting data of interest from Web documents.

XML is actually a subset of Standard Generalized Markup Language [SGML], or SGML, that is designed to be served, received, and processed in the Web. From this document-centric perspective, XML is perceived as a physical representation (data format). However, from a data-centric perspective XML is seen as a representation of information at logical level, which embeds metadata (schema) in data. From this point of view, the focus is on querying data represented in XML. XML provides a suitable framework for the convergence of the document-centric and the data-centric perceptions of the Web [ABS00].

2.3.1 XML Syntax and DTD

Each XML document has a *logical structure* and a *physical structure*. Physically, the document is composed of units called *entities*. An entity may refer to other entities to cause their inclusion in the document. A document begins with a "root" or *document entity*. Entities contain *parsed data* or *unparsed data*. Parsed data is made up of characters, some of which form *character data*, and some of which form *markup*. Markup encodes a description of the storage layout and the logical structure of the document. The logical structure includes *declarations*, *elements*, *comments*, *character references*, and *processing instructions*. Following a data-centric perspective, in what follows we focus on the logical structure of XML documents, and specifically on declarations and elements. Example 2.2 lists part of an XML document that represents the OEM of Figure 2.1.

Example 2.2

```
<database id="db">
    <biblio>
```

```
        <book id="n1">
            <author>Roux</author>
            <author>Combalusier</author>
            <date>1976</date>
            <title>Database Systems</title>
        </book>
        <book id="n2">
            <author>Smith</author>
            <date>1999</date>
            <title>Database Systems</title>
        </book>
        <paper>...</paper>
    </biblio>
</database>
```

♦

An XML document contains one or more *elements*, delimited by *start-tags* and *end-tags*, or by *empty-element-tags* for *empty elements*. The *content* of an element are the characters between the start-tag and the end-tag. Elements have a *type*, identified by the element name, and a set of *attribute* specifications. Each attribute specification has a name and a value. An *element declaration* constrains the type and the content of the element. The predefined types are: "`EMPTY`", "`ANY`", `mixed`, and `children`. In case of `children`, the element type has *element content*, meaning that only child elements are contained and no free character data. In this case, the constraints include a *content model*, a simple grammar denoting the allowed types of child elements and the order in which they are allowed to appear[4]. The grammar is built on choice lists or sequence lists of content particles. In case an element type has `mixed` content, the types of the child elements may be constrained, but not their order or their number of occurrences. *Attribute-list declarations* may be used to: (a) define a set of attributes for a given element type, (b) establish type constraints for these attributes, and (c) provide default values. The possible *attribute types* are: `StringType`, which is marked "`CDATA`" and denotes that the attribute value is a sequence of characters, `EnumeratedType`, and `TokenizedType`. In the case of `TokenizedType`, the attribute can be declared among other things as: "`ID`", which implies a unique identifier for the corresponding element, "`IDREF`", which is used to reference another element using its "`ID`" attribute, and "`IDREFS`", which is used to reference multiple elements at once using their "`ID`" attributes. Attributes of type "`ID`" together with "`IDREF`" and "`IDREFS`" can be used to convert the XML document tree into a graph, by allowing an element to be referenced by more than one "parent" elements. The *default value* of an attribute may optionally contain a keyword and a value, as in "`#REQUIRED`", "`#IMPLIED`", "`#FIXED <def_value>`", or just "`<def_value>`".

An XML document is said to be *well-formed* if:

1. It matches the production labeled "`document`" in [XML], which defines the grammar of XML using EBNF notation.
2. All references within the document are defined.

Matching the "`document`" production implies that all elements rest within each other, and that there is a single element called *root* or *document element*, which is not in the content of any other element.

[4] Order constitutes a difference between XML and semistructured data models seen in the previous section. Because of the document origins of XML, the order of elements is important. On the other hand, the order of objects is not important in the semistructured data models of Section 2.2.1.

Example 2.3

```
<!DOCTYPE database [
    <!ELEMENT biblio (book*, paper*)>
    <!ELEMENT book   (author+, date, title, abstract?)>
    <!ELEMENT paper  (author+, date, title, published, abstract?)>
    <!ELEMENT author    #PCDATA>
    <!ELEMENT date      #PCDATA>
    <!ELEMENT title     #PCDATA>
    <!ELEMENT published #PCDATA>
    <!ELEMENT abstract  #PCDATA>
    <!ATTLIST database id ID #REQUIRED>
    <!ATTLIST book     id ID #REQUIRED>
]>
```

♦

To define constraints on the logical structure and on the storage layout of XML documents, Document Type Definition, or DTD, can be used. Example 2.3 shows a DTD for the XML document of Example 2.2. A DTD contains *markup declarations*, which can be *element type declarations*, *attribute-list declarations*, *entity declarations*, and *notation declarations*. Element type declarations and attribute-list declarations are used to define constraints on the types and contents of XML elements and attributes. An XML document is *valid* with respect to a DTD, if it is associated with that DTD, and if it complies with the constraints expressed in the DTD. The DTD section must appear before the first element, and can (a) contain markup declarations directly inside the XML document, (b) point to an external document containing markup declarations, or (c) do both, in which case the DTD of the document consists of both declaration subsets taken together.

2.3.2 XML–Related Technologies

The World Wide Web Consortium [W3C] is responsible for controlling and promoting XML issues, and has published a number of specifications on related technologies that exploit and advance XML. In this section we overview briefly some of those specifications, which seem to be have an important place in the picture of a rich Web infrastructure.

DTDs are limited in what they can express. For instance, we cannot use a DTD to specify that the element `price` should be a fixed-precision real number, or that an attribute of type "`IDREF`" should only point to elements of specific type. For this reason, XML Schema [XSC0, XSC1, XSC2] was proposed, which incorporate features that cover such requirements. XML Schema includes a number of predefined basic types that can be used for defining new data types. Data types may be complex types that are based on other data types. An XML Schema is itself an XML document.

Like DTD, XML Schema describes the logical structure of XML data; it does not address the issue of information semantics. The issue of semantics is important because knowing what a piece of data is about and how it relates to other data will allow to build better search engines and more advanced applications. The Resource Description Framework [RDF1, RDF2], or RDF, provides a general method that resembles an E-R diagram to describe semantic metadata for XML documents. It describes *resources*, identified by URIs [URI], which have *properties*. *RDF statements* are triads, composed by a *subject* (resource), a *predicate* (property), and an object (property value), therefore associating property-value pairs with resources. The value of a property may be a literal, or another resource. RDF statements are themselves expressed in XML.

Namespaces [XNS] enable a single XML document to contain *markup vocabulary* defined in a number of different DTDs. Markup vocabulary consists of element names and attribute names. Since the same name may be declared in two different DTDs with different meaning and type, mixing those two DTDs may result in a collision. Namespaces solve this collision problem by introducing a global prefix that precedes element and attribute names.

XLink [XLIN] define links between XML documents, document parts, or resources in general. Those links are more powerful that HTML links, and are able to assert relationships between multiple resources. Moreover, it is possible to attach metadata to a link, describing attributes of the link itself. XLink is expressed in XML, and may reside within or outside the related documents. Resources are identified with URIs, except if we want to refer to a part inside an XML document, in which case the XPointer [XPTR] specification can be used.

As mentioned before, XML restricts itself to describing the physical and logical structure of documents and does not deal with presentation issues. Presentation is addressed by XML Stylesheets [XSL], or XSL, and XSL Transformations [XSLT], or XSLT. XSL is a language for formatting XML, while XSLT is part of XSL and is a language for transforming XML documents. An XSLT document is a collection of transformation rules that operate on a source XML document to produce a new document. The result document may be an XML document, an HTML document, or a document in some other format, depending on the transformation rules. Each rule consists of a *pattern* and a *template*. Starting from the root, patterns are matched recursively against the source tree and the template is instantiated to produce part of the result tree. Parts of the source tree that are not matched are simply copied to the result tree. XSL and XSLT are expressed in XML.

Example 2.4

```
<xsl:template match="/">
    <html>
        <head>
            <title>Book Titles</title>
        </head>
        <body>
            <h1>Book Titles</h1>
            <xsl:for-each select="database/biblio/book">
                <xsl:value-of select="title"/>
                <br></br>
            </xsl:for-each>
        </body>
    </html>
</xsl:template>
```

♦

Example 2.4 shows part of an XSLT document that transforms the XML in Example 2.2 to an HTML document listing the titles of books. Notice the use of XML Namespaces and the global prefix `xsl` that precedes elements defined as part of XSLT. XSL and in particular its transformation part, XSLT, can be considered as a limited query language for XML [ABS00, BC00].

In the XSLT of Example 2.4, `database/biblio/book` is the abbreviated form of an XPath [XPTH]. An XPath is similar to SSD path expressions, starts from a node called *context node*[5] and continues with a sequence of *steps*. Each step has an *axis*, which gives navigational directions, a node test, and an optional list of *predicates*. As an example consider the XPath

[5] In the frame of the present work we use the term **context node** with a completely different meaning. In Chapter 4 we introduce context node as a special type of node that represents facets of **multidimensional entities**.

`root()/descendant::book/child::author[position()=first()]`, which returns the first authors of books in the XML of Example 2.2. In this XPath, `descendant` and `child` are axes, `book` and `author` are node tests matched against element names, and `[position()=first()]` is a predicate that is satisfied only by the first author of each book.

2.3.3 XML Query Languages

As mentioned in the previous section, XSL can be seen as an XML query language. However, the expressiveness of XSL as a query language is limited [ABS00, BC00]. The importance of a powerful XML query language was soon realized [Abi99], and a number of objectives that such a language should fulfill were compiled [Mai98, DFF+99, XQR].

Because of the strong resemblance between XML and semistructured data models, OEM and Lorel were adapted [GMW99] to XML without great difficulty. The main changes have to do with incorporating features for handling attributes and attribute references. A Lorel query can treat XML attribute references in a dual way: either as plain attributes, or as edges pointing to "subelements". Other additions are *range qualifiers* and an *order-by* clause that address the ordering of elements in XML documents, and the use of Skolem functions in the construction of results to avoid multiple creations of the same element.

In contrast to Lorel, XML-QL [DFF+99a, DFF+99] is a query language designed directly for XML. The data model of XML-QL is XML Graph, a variant of the familiar edge-labeled directed graph, which comes in two flavors: an unordered and an ordered one. Similarly to UnQL, XML-QL uses a `where` clause for binding variables in *element patterns*, and a `construct` clause for creating the resulting XML document. Moreover, XML-QL supports *tag variables* that are similar to label variables of UnQL, and *regular path expressions* and *wildcards* for formulating more powerful element patterns.

```
<Smith_publ>
    WHERE <$p>
              <author>Smith</>
              <date> $d </>
              <title> $t </>
          </> IN "www.site.edu/biblio.xml",
          $p IN {book, paper}
    CONSTRUCT <$p>
                  <title> $t </>
                  <date> $d </>
              </>
</Smith_publ>
```

The element pattern in the above XML-QL query matches `book` and `paper` elements in the document `biblio.xml` (see Example 2.2), which contain an `author` subelement with value "Smith". The tag variable `$p` transfers element names (`book` or `paper`) from the input document to the newly constructed elements. In the result document, constructed elements are subelements of the root element `Smith_publ`.

Other query languages for XML are XML-GL [CCD+99], a graphical XML query language, YATL [CS00], which is based on functional programming languages, and XQL [XQL], a language that extends XSL. Comparisons between XML query languages can be found in [BC00, FSW99].

Quilt [CRF00] is an XML query language that borrows features from many others, namely XPath, XQL, XML-QL, SQL [SQL], OQL [ODMG], Lorel, and YATL. The structure of Quilt is

based on FLWR (pronounced "flower") expressions, which may include `for`, `let`, `where`, and `return` clauses. Variables are bound in `for` clauses, `let` clauses provide bindings for additional variables, `where` clauses filter the tuple bindings generated by `for` and `let`, and `return` clauses construct the results. The following Quilt query lists the book authors of Example 2.2 in alphabetical order, and for each author his or her books ordered by publication date:

```
<author_list>
    FOR $a IN distinct(document("biblio.xml")//author)
    RETURN
        <author>
            <name> $a/text() </name>
            FOR $b IN document("biblio.xml")//book[author=$a]
            RETURN
                <book>
                    $b/title,
                    $b/date
                </book> SORTBY date DESCENDING
        </author> SORTBY name
</author_list>
```

Quilt uses XPath for navigating in XML documents. For instance, the XPath `document("biblio.xml")//book[author=$a]` matches all `book` elements in `biblio.xml` that have an `author` subelement with value equal to the value of variable `$a`. Moreover, the XPath `$b/title` causes `title` subelements of books bound to `$b` to be inserted in the results. Note how Quilt allows nesting of expressions in a general and flexible way. Quilt forms the basis of XQuery [XQRY], an XML query language endorsed by W3C.

2.4 MSSD AND MXML

In this thesis we incorporate context in semistructured data and define **multidimensional semistructured data** (**MSSD**). In particular, we extend OEM and propose **Multidimensional Data Graph** and **Multidimensional OEM** (**MOEM**), a data model for MSSD. To represent MSSD textually we introduce **mssd-expressions**, an extension of ssd-expressions. We also present **Multidimensional Query Language** (**MQL**), which integrates context with features from Lorel and UnQL. Moreover, we extend XML, DTD, and XSL, and define **Multidimensional XML** (**MXML**), **MDTD**, and **MXSL**, in which context plays a principal role.

Although the notion of context is not supported as part of XML, its importance is recognized indirectly. XML defines the special attribute `xml:lang`, which may be inserted in XML documents to specify the language used in the contents and the attribute values of any element. In the following chapter we review previous work on context, and present our own approach.

3 A FORMALISM FOR CONTEXT

Multidimensional semistructured data (MSSD for short) represent information entities that assume different facets, with varying content (value and / or structure), under different contexts. This chapter introduces a formalism for expressing context through variables called **dimensions**, and establishes a correspondence between context and *possible worlds*. In addition, a number of context operations are defined, and their properties are investigated. Finally, a convenient notation is proposed for context expressions and context conditions.

The approach presented in this chapter sees context as a set of constraints, which are combined to specify environments under which information obtains an unambiguous meaning. Such an environment is called a **world**, while constraints are expressed by assigning values to dimension variables. Context, as defined in this chapter, aims to be flexible and to take into account the special characteristics and dynamic nature of Web data. In Web applications, producers and consumers of context-dependent information may not have the same idea about what context should cover. Therefore, information producers should not be obliged to create constraints for every possible dimension, while information consumers should be able to shift constraints to their own frame of interpretation, which possibly means the use of additional dimensions. Being able to deal with new dimensions without changing the meaning of existing context expressions would also be desirable when integrating heterogeneous context-dependent information from various Web sources.

3.1 RELATED WORK

In the past decade the notion of context has been applied in various fields of computer science. An extensive bibliography review that appears in [The01] discusses the use of context in linguistics, music, software development, machine learning, networks, artificial intelligence, semantic model clustering, nested associations, categorization, information bases, and the Web. Specifically for the Web, where "authors of Web pages have diverse backgrounds, knowledge, culture, and aims" and "the availability of metadata is inconsistent", context is not exploited to its full potential and "greater use of context in web may help increase competition and diversity on the web".

Broadly speaking, context is used as a tool for reasoning with viewpoints and background beliefs, and as an abstraction mechanism that helps dealing with complexity, heterogeneity, and partial knowledge. In what follows, we focus on a number of approaches that are interesting from the perspective of the present work.

Context is examined in [Bun94] from a linguistic point of view, and is perceived as depending on a set of *factors*, grouped into general categories, which are also called *dimensions*. A distinction is made between *global aspects*, which tend to remain constant, and *local aspects*, whose value change dynamically depending on the feedback of the application in which context is used. In our approach we define context by assigning specific values to

dimensions. Therefore, although we apply context in a different frame, we could say that our notion of context addresses what is described in [Bun94] as global aspect factors.

In [Giu93] context is used as a means of formalizing reasoning upon only a subset of a global knowledge base, an approach called *localization*. *Local Model Semantics* that are proposed in [GG01] formalize two principles: the *principle of locality*, according to which reasoning uses only part of what is potentially available and this part forms the context, and the *principle of compatibility*, which states that there exists a compatibility between reasoning performed in different contexts. Local model semantics can be applied to reasoning with viewpoints; the principle of locality deals with having different views of the same entity, while the principle of compatibility relates those views in consistent constructs.

As stated in [OS99], context can be used in global information systems to capture the *real-world semantics* of an object. Moreover, the use of context may lead to several benefits.

- *Economy of representation*: Context can act as mechanisms for isolating and accessing parts of information, in a way similar to database views.
- *Economy of reasoning*: Reasoning can be performed with the context associated with a single information source instead of the whole global database.
- *Managing inconsistent information*: When dealing with integrating independent information sources, inconsistency of information may be allowed as long as information remains consistent within the context of user queries.
- *Flexible semantics*: The relation and the semantic proximity between two objects may vary according to the context.

The formalism of Local Model Semantics [GG01] is used in [GS98a] to represent a federated database as a set of local models that correspond to the different databases of the federated system. The same formalism is applied in [GS98b] for integrating information from a number of autonomous agents operating in the field of electronic commerce. In [FDFP95] an extension of first-order logic called context logic is used, for integrating heterogeneous information sources incrementally. For each source, the *information source context* holds the original schema, and the *semantic context* expresses implicit assumptions about the schema and contains translation rules. Finally, the *integrating context* gathers rules from several semantic contexts and other integrating contexts, and provides a unified model that deals with certain forms of heterogeneity.

In [MM95] context provides a generic viewpoint mechanism used to partition an information base in manageable fragments containing related objects. This idea is further developed in [The01, TACS98], where contexts and operations between contexts are formally defined. A context is associated to a *lexicon*, which in turn associates global object identifiers to a set of alternative names through which objects can be referenced. The same object may have different names in different contexts. Objects in a context can be contexts themselves, leading to arbitrarily deep nested structures. Operations defined include *union*, *intersection*, and *difference* between contexts. *Configuration contexts* represent complex objects composed by particular versions of their components, and can be used to coordinate cooperation tasks where many persons work on the same documents.

In this chapter we define operations between contexts similar to [The01], like for example **context union**, **context intersection**, and **context difference**. In addition, in Chapter 4 we specify **reduction** processes, which correspond to viewpoint mechanisms and project part of the available information depending on context. However, our view of what context is and how it is represented and used differs fundamentally. In [MM95, The01, TACS98] contexts are themselves objects, acting as containers of other objects that are relevant in a particular real-

world situation. In our approach context is metadata attached to objects and to relations between objects, and provides a framework for the interpretation of data.

In this interpretation framework, context represents a set of *possible worlds*. In Modal Logics [Fit93] possible worlds qualify the truth values of propositions. For instance, the sentence "I see that it is raining" is false at the world in which I am in Athens and the date is 3 March 2003, but it may be true at a world where I am in another place at another time. Analogously, worlds in MSSD qualify information entities, and define which variants of an information entity hold under some given conditions and how those variants are to be interpreted.

Worlds in MSSD are expressed in an intuitive way, using (*name*, *value*) pairs of variables called dimensions. Similar constructs have been used independently in a couple of occasions. In [NP03] (*name*, *value*) pairs of *characteristics* such as `language` are used to parameterise the process of automatic Web publishing. In addition, dimensions have been used in Intensional HTML [WBSY98, Bro98, Yil97, Bro98a], or IHTML in short, a Web authoring language that is based on and extends ideas proposed for a software versioning system [PW93]. IHTML allows a single Web page to have different variants and to dynamically adapt itself to a given world. Those worlds are specified by assigning a single value to each dimension. IHTML follows a document-centric approach, using a naming scheme to encode worlds in the filenames of document versions. Our work was influenced by IHTML, but follows a different direction: (a) context represents a *set* of possible worlds, and dimensions used to express a context can be assigned many values and be combined in conjunctions and disjunctions, (b) we formalize the notion of context and introduce operations between contexts, and (c) we focus on a data-centric approach, which aims to incorporate context in data models and query languages for semistructured data.

3.2 CONTEXT, DIMENSIONS, AND WORLDS IN MSSD

The main difference between multidimensional and conventional semistructured data lies in the concept of **multidimensional entity**. A multidimensional entity is an information entity that assumes different facets under different conditions. As an example, consider the case of the price of a book on the Internet, expressed both in Euros and US Dollars: `price` is a multidimensional entity with value `25` if the currency is Euro, and `22` if the currency is US Dollar[6]. The two values are two facets of the same `price` entity, while currency is represented by a **dimension** variable that defines the circumstances under which each facet holds – in other words, the *context* under which each facet holds.

In the `price` example, context is formed using just one dimension, namely `currency`. In the general case, however, any number of dimensions can participate in formulating a context. As an example, consider information in a document that must exist in a number of different languages, in a number of physical formats, and in varying degrees of detail: one dimension is `language` that can take the values `english`, `spanish`, and `greek`; another dimension is `format` with possible values `pdf`, and `ps`; finally, the dimension `detail` can be either `low`, `medium`, or `high`. Contexts are formed by combining conditions that are based on assigning values to dimensions: one facet of the `document` multidimensional entity can hold under low or medium detail, pdf format, and English language, while another facet can hold under low detail, pdf format, and Greek language.

[6] The exchange rate at the time of writing this example was 1 Euro for 0.88 US Dollars.

Context is used not only to define the exact conditions under which various facets hold, but also to express requirements against which a multidimensional entity is evaluated and some of its facets are selected. For instance, the requirements that detail must be low and that language must be either English or Greek, constitute a context that can be matched[7] against the previously mentioned facets of the `document` multidimensional entity.

3.2.1 Assumptions about Dimensions

For the rest of this chapter we make a number of assumptions about dimensions and dimension domains:

1. Dimension names and dimension values in respective domains have a global scope, and their meaning is the same for everybody.
2. Dimensions[8] are *orthogonal*, in other words, the value assigned to one does not affect the value of the others.
3. We only consider sets of dimensions **D** that are nonempty and finite.
4. Dimension domains are also assumed to be nonempty and finite, and it must be possible to describe them by enumerating their elements.

Other ways of representation, as well as infinite domains, may be useful and are not excluded, they are however outside of the scope of the present work. The same is true for dimensions that are not orthogonal and have interrelated domains.

Context is always interpreted with respect to a set of dimensions **D** and the relevant dimension domains. In addition, in order to carry out certain operations on context we must take into account the dimensions in **D** and the dimension domains, $\mathbf{V}_d$, $\forall$ d $\in$ **D**. Dependence on **D** will be examined when defining a relevant operation, and the symbol $_D$ will be used adjacent to the operation symbol if an operation depends on **D**. In what follows, the phrase "with respect to **D**" ("w.r.t. **D**" in short) declares dependence on a set of dimensions **D**.

An important point is that the set of dimensions **D** considered for a given problem may vary from case to case. On the contrary, the domain $\mathbf{V}_d$ of a dimension d is assumed in any case invariable.

In the frame of a single problem, *the set of dimensions* **D** *must contain at least those dimensions that participate in the contexts involved. We use* $\mathbf{D}_{min}$ *to represent the set containing only those dimensions that appear in the contexts of a given problem*.

3.2.2 Possible Worlds

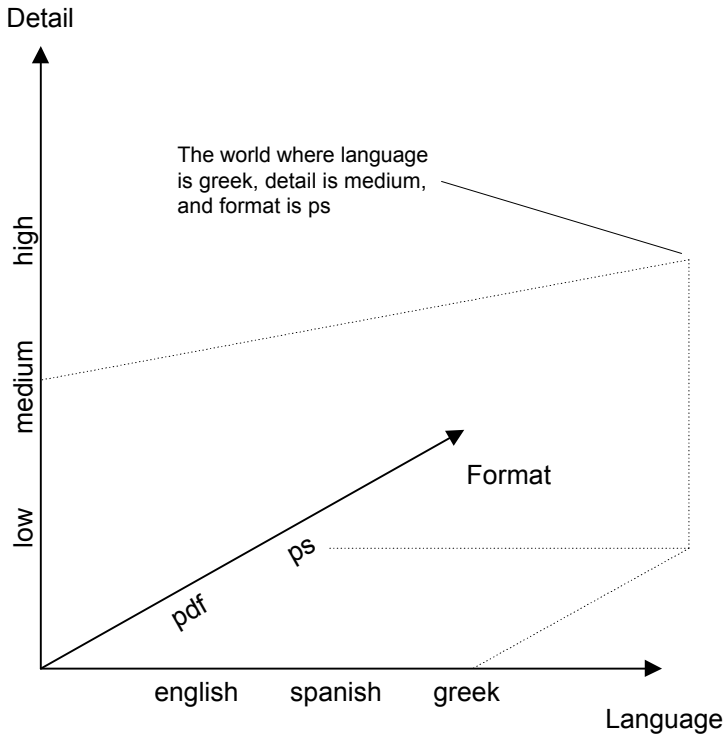
An alternative way to view context is through the notion of **world**, which is fundamental in MSSD. A world represents an environment under which data obtain a substance, and

[7] Explained in detail in the following chapters.

[8] The term **dimension** is also used in Data Warehousing [CD97, Coll96] to describe data that qualify pieces of information. Dimensions in Data Warehousing are not always considered orthogonal and their domains are often organized in a tree-like manner: consider for instance a dimension `continent` with values `america`, `europe`, `asia`, `africa`, `australia`, and a dimension `country` with values the countries in every continent. The difference between the notions of dimension in Data Warehousing and in MSSD is that, in Data Warehouses dimensions are used mainly to navigate and to aggregate data, whereas in MSSD dimensions are used to provide an environment for the interpretation of data.

provides complete and specific guidance for the interpretation of information. If we consider dimensions and their discrete domains as defining a multidimensional space, then each point in that space represents a possible world, as depicted in Figure 3.1.

Figure 3.1: Point in multidimensional space representing a possible world.



A world, therefore, can be defined by assigning a single value to every dimension, as it is stated formally in the following definition.

Definition 3.1

Let $\mathbf{D}$ *be a nonempty set of dimension names, and for each* $d \in \mathbf{D}$ *let* $\mathbf{V}_d$ *be the domain of d, with* $\mathbf{V}_d \neq \varnothing$*. A* ***world*** *w with respect to* $\mathbf{D}$ *is a set of pairs (d,v), where* $d \in \mathbf{D}$ *and* $v \in \mathbf{V}_d$*, such that* $\forall\, d \in \mathbf{D}$ *exactly one (d,v) belongs to w.*

For determining whether a set of *(dimension, value)* pairs is actually a world, the set of dimensions **D** must be taken into account. A set of pairs that represents a world with respect to **D**, does not represent a world with respect to any other set **D′**.

As an example, consider the sets $\mathbf{D}_1$={`language, format`} and $\mathbf{D}_2$={`language, format, detail`}, with domains as shown in Figure 3.1. The set {`(language,greek), (format,pdf)`} is a world with respect to $\mathbf{D}_1$, while the set {`(language,greek), (format,pdf), (detail,medium)`} is a world with respect to $\mathbf{D}_2$.

The set of all possible worlds with respect to a set of dimensions **D** is called the **universe of D**. For instance, the universe of $\mathbf{D}_1$ is the set:

```
{{(language,english),(format,pdf)},
{(language,english),(format,ps)}, {(language,spanish),(format,pdf)},
{(language,spanish),(format,ps)}, {(language,greek),(format,pdf)},
{(language,greek),(format,ps)}}
```

Definition 3.2

*The **universe of D**, denoted U_D, is the set of all possible worlds with respect to **D**.*

The notion of world gives an insight to context, which can be seen as defining a set of worlds in a compact way. In the following section we explain how context is expressed in terms of dimensions, and the operations that can be performed on contexts – including operations that transform a context into the set of worlds it represents.

3.3 CONTEXT SPECIFIERS AND CONTEXT OPERATIONS

Context in MSSD is expressed through **context specifiers**, which are syntactic constructs involving dimensions and dimension values. Context specifiers qualify semistructured data pieces, defining the worlds under which those pieces hold. Thus, context specifiers are the most important difference between multidimensional and conventional semistructured data.

Since a context specifier can be interpreted as a set of worlds, the basic operations between context specifiers are similar to conventional set operations. In this section we define **context intersection**, **context union**, **context difference**, **context equality**, and **context superset / subset**. Moreover, the operations of **context expansion** and **context extension** are used to obtain the actual set of worlds represented by a context specifier. Additional operations on context specifiers, like for example the cartesian product of two context specifiers, could also be defined. In this section, we do not address every possible context operation, but we focus on operations that will be useful in subsequent chapters. Additional operations like the cartesian product are not excluded, but the need for such operations has not come up in the frame of the present thesis.

A context specifier is a complex structure based on a simpler structure, called **context specifier clause**. We first discuss context specifier clauses and context operations on them, and then proceed to examine context specifiers with their operations.

3.3.1 Context Specifier Clauses

Context specifier clause[9] (or just **clause**) is based on a structure called **dimension specifier**. A dimension specifier can be seen as a way to express constraints on the value of a single dimension. As an example, consider the proposition "the value of dimension `language` is `english` or `spanish`"; the corresponding dimension specifier is `(language, {english, spanish})`. A conjunction of such constraints, where each concerns a different dimension, forms a context specifier clause. The context specifier clause `{(language,{greek}), (detail,{low,medium})}` defines the worlds where language is Greek *and* detail is either low or medium.

[9] This structure is called *context specifier clause* because it resembles a Horn Clause [Ull88, Kel97]: $p_1 \wedge \ldots \wedge p_n \rightarrow q$, where p_i correspond to dimension specifiers and q to the corresponding facet of the multidimensional entity.

Definition 3.3

Let $\mathbf{D}$ *be a nonempty set of dimension names, and for each* $d \in \mathbf{D}$ *let* $\mathbf{V}_d$ *be the domain of d, with* $\mathbf{V}_d \neq \emptyset$. *A* ***dimension specifier*** *s for a dimension d is a pair (d,V) where* $d \in \mathbf{D}$ *and* $V \in 2^{\mathbf{V}_d}$, *the power set of* $\mathbf{V}_d$. *A* ***context specifier clause*** c^{cl} *is a set of dimension specifiers, such that for any dimension* $d \in \mathbf{D}$ *there exists at most one dimension specifier (d,V) in* c^{cl}.

Any number of dimensions can be used to form a context specifier clause; in other words, it is not necessary for a context specifier clause to contain an element (dimension specifier) for every possible dimension. Omitting dimensions is a convenient shortcut whose meaning will become clear when we show how a context specifier clause is transformed to a set of worlds.

In what follows, the symbol ^{cl}, standing for *clause*, will be used to identify context specifier clause variables and operations. This convention will be useful later for distinguishing clause variables and operations, from context specifier variables and operations.

3.3.1.1 Correspondence to Worlds

The correspondence between a context specifier clause and the set of worlds that it represents is defined in what follows.

Definition 3.4

Let $\mathbf{D}$ *be a nonempty set of dimension names, and for each* $d \in \mathbf{D}$ *let* $\mathbf{V}_d$ *be the domain of d, with* $\mathbf{V}_d \neq \emptyset$. *Let* c^{cl} *be a context specifier clause, and w be a world with respect to* $\mathbf{D}$. *Then, w is represented by* c^{cl} *iff for every pair* $(d,v) \in w$, *either* $(d,V) \in c^{cl}$ *with* $v \in V$, *or there exists no pair (d,V) in* c^{cl}.

The set of worlds represented by a context specifier clause can be obtained using the operations of clause expansion and clause extension, defined in what follows.

Clause Expanded Form

Clause expansion $\odot^{cl}_D$ is a unary operation that takes a context specifier clause c^{cl}_1 and returns a context specifier clause c^{cl}_2. Clause expansion is performed with respect to a set of dimensions $\mathbf{D}$ and to the sets of dimension domains $\mathbf{V}_d$ for each d in $\mathbf{D}$.

$\odot^{cl}_D : C^{cl} \rightarrow C^{cl}$, where C^{cl} is the set of all clauses

The resulting clause c^{cl}_2 is called the **expanded form** of clause c^{cl}_1 with respect to $\mathbf{D}$, and is defined as follows:

Definition 3.5

Let $\mathbf{D}$ *be a nonempty set of dimension names, and for each* $d \in \mathbf{D}$ *let* $\mathbf{V}_d$ *be the domain of d, with* $\mathbf{V}_d \neq \emptyset$. *Let* c^{cl} *be a context specifier clause. Then the* ***clause expansion*** *of* c^{cl} *with respect to* $\mathbf{D}$ *(also called* ***expanded form*** *of* c^{cl} *w.r.t.* $\mathbf{D}$*), denoted* $\odot^{cl}_D\ c^{cl}$, *is a context specifier clause containing elements as follows: (a) if* $(d,V) \in c^{cl}$ *then* $(d,V) \in \odot^{cl}_D\ c^{cl}$, *and (b) if* $d \in \mathbf{D}$ *and there is no pair (d,V) such that* $(d,V) \in c^{cl}$ *then* $(d,\mathbf{V}_d) \in \odot^{cl}_D\ c^{cl}$.

The expanded form of a context specifier clause is *equivalent to that clause with respect to* $\mathbf{D}$, in the sense that they both represent the same set of worlds with respect to $\mathbf{D}$.

Proposition 3.1

A context specifier clause and its expanded form with respect to $\mathbf{D}$ *represent the same set of worlds w.r.t.* $\mathbf{D}$*, for any set of dimensions* $\mathbf{D}$*.*

Proof: The expanded form w.r.t. $\mathbf{D}$ may contain additional constraints in the form of dimension specifiers $(d,\mathbf{V}_d)$. However, such dimension specifiers do not effectively constraint the possible values of a dimension d, since d may range over its whole domain $\mathbf{V}_d$. Therefore, according to Definition 3.4, they do not cause the exclusion of any world w.r.t. $\mathbf{D}$.

The role of omitted dimensions is now obvious: for every dimension in $\mathbf{D}$ that does not have a corresponding dimension specifier in a clause, the inclusion in that clause of a dimension specifier containing all values in the dimension domain is implied. Dimension specifiers in the expanded form of a clause have always a one-to-one correspondence[10] with dimensions in $\mathbf{D}$. It is obvious that the same context specifier clause will have different expanded forms with respect to different sets of dimensions.

Example 3.1

Let:

$\mathbf{D}$ = {lang, detail, format}
$\mathbf{V}_{lang}$ = {en, gr, sp}
$\mathbf{V}_{detail}$ = {low, med, high}
$\mathbf{V}_{format}$ = {pdf, ps}

c^{cl}_1 = {(lang,{en,sp}), (format,{pdf})}
c^{cl}_2 = {(lang,{en}), (detail,{med,high}), (format,{ps})}
c^{cl}_3 = {(lang,{gr}), (format,{})}
c^{cl}_4 = {}

Then, the expanded forms with respect to $\mathbf{D}$ are:

$\odot^{cl}_D\ c^{cl}_1$ = {(lang,{en,sp}),(format,{pdf}),(detail,{low,med,high})}
$\odot^{cl}_D\ c^{cl}_2$ = {(lang,{en}),(detail,{med,high}),(format,{ps})}
$\odot^{cl}_D\ c^{cl}_3$ = {(lang,{gr}),(format,{}),(detail,{low,med,high})}
$\odot^{cl}_D\ c^{cl}_4$ =
{(lang,{en,gr,sp}),(detail,{low,med,high}),(format,{pdf,ps})}
♦

In Example 3.1, $\odot^{cl}_D\ c^{cl}_2$ is the same as c^{cl}_2, since this clause contains dimension specifiers for every dimension in $\mathbf{D}$. The empty set is a legal context specifier clause, and its expanded form covers the complete domains of all dimensions in $\mathbf{D}$, as is shown by c^{cl}_4 in Example 3.1. Notice that *the expanded form of a clause can never be the empty set*, since the expansion will add dimension specifiers for every d in $\mathbf{D}$.

Clause Extension

Clause extension $\times^{cl}$ is a unary operation that takes a context specifier clause c^{cl}, with $c^{cl} \neq \emptyset$, and returns a set of sets of *(dimension, value)* pairs.

$\times^{cl}$: $(C^{cl}-\emptyset) \rightarrow$ S, where C^{cl} is the set of all clauses, and
S contains sets of sets of *(d,v)* pairs

Clause extension is defined below.

[10] As stated in Definition 3.3, if (d,V) belongs to a context specifier clause, then d must belong to the set of dimensions $\mathbf{D}$.

Definition 3.6

*The **clause extension** of a context specifier clause $c^{cl} = \{(d_1,V_1), (d_2,V_2),..., (d_n,V_n)\}$, with $n \geq 1$, is denoted $\times^{cl} c^{cl}$, and is a set defined as follows: if $(d,\varnothing) \in c^{cl}$, then $\times^{cl} c^{cl} = \varnothing$, else $\times^{cl} c^{cl} = \{ \{(d_1,v_1), (d_2,v_2),..., (d_n,v_n)\} \mid v_i \in V_i$, with $1 \leq i \leq n \}$.*

Therefore, the extension of a clause c^{cl} is a set of sets of *(dimension, value)* pairs. To determine whether such a set of *(dimension, value)* pairs represents a world, according to Definition 3.1 the set of dimensions **D** must be taken into account. It is easy to see that, if dimension specifiers (d,V) in c^{cl} have a one-to-one correspondence with dimensions in **D**, then $\times^{cl}\ c^{cl}$ gives the set of worlds w.r.t. **D** represented by c^{cl}. Clause expansion, however, guarantees this condition; this fact together with Definition 3.4 and Proposition 3.1 leads to the following:

Proposition 3.2

*The set of worlds w.r.t. **D** represented by a context specifier clause c^{cl}, denoted $W_D(c^{cl})$, is given by the clause extension of the expanded form w.r.t. **D** of c^{cl}.*

$$W_D(c^{cl}) = \times^{cl} (\odot^{cl}_D\ c^{cl}) \qquad \textbf{Formula 3.1}$$

Proof: According to Proposition 3.1, $W_D(c^{cl}) = W_D(\odot^{cl}_D\ c^{cl})$. We must show that $W_D(\odot^{cl}_D\ c^{cl}) = \times^{cl} (\odot^{cl}_D\ c^{cl})$. Since **D** is nonempty, the result of expansion can never be the empty set, therefore extension can always take place. It is easy to see that $\times^{cl} (\odot^{cl}_D\ c^{cl})$ produces worlds w.r.t. **D**, which are in line with the constraints (dimension specifiers) in c^{cl}, as specified in Definition 3.4. In addition, there cannot be a world w.r.t. **D** which conforms to the constraints in c^{cl} and is not given by $\times^{cl} (\odot^{cl}_D\ c^{cl})$.

Example 3.2

Let:

$\mathbf{D} = \{lang, detail\}$
$\mathbf{V}_{lang} = \{en, gr, sp\}$
$\mathbf{V}_{detail} = \{low, high\}$

$c^{cl}_1 = \{(lang,\{en,gr\})\}$
$c^{cl}_2 = \{(lang,\{en\}), (detail,\{low,high\})\}$
$c^{cl}_3 = \{(detail,\{\})\}$

Then, the following hold:

1. $\times^{cl}\ c^{cl}_1 = \{\{(lang,en)\}, \{(lang,gr)\}\}$
   $\times^{cl} (\odot^{cl}_D\ c^{cl}_1) =$
   $\{ \{(lang,en),(detail,low)\}, \{(lang,en),(detail,high)\},$
   $\{(lang,gr),(detail,low)\}, \{(lang,gr),(detail,high)\} \}$

2. $\times^{cl}\ c^{cl}_2 = \times^{cl} (\odot^{cl}_D\ c^{cl}_2) =$
   $\{ \{(lang,en),(detail,low)\}, \{(lang,en),(detail,high)\} \}$

3. $\times^{cl}\ c^{cl}_3 = \times^{cl} (\odot^{cl}_D\ c^{cl}_3) = \varnothing$

♦

In contrast to case (1) of Example 3.2, where the introduction of expansion changed the results of clause extension, in case (2) the extension of c^{cl}_2 is equal to the extension of the

expanded form of c^{cl}_2. In case (1) of Example 3.2, the extension $\times^{cl}$ c^{cl}_1 does not give worlds w.r.t. **D**, because there are not any pairs for the dimension `detail`. Note however, that if the set of dimensions **D** was equal to {`lang`}, then $\times^{cl}$ c^{cl}_1 would give worlds w.r.t. **D**. This observation can be generalized: *a context specifier clause that represents a single world w.r.t.* **D**, *represents a (nonempty) set of worlds w.r.t. any* **D′** $\supset$ **D**.

Clause Expanded Extension

Expansion and extension are often used together to give the set of worlds w.r.t. **D** represented by a clause. An operation that combines expansion and extension is introduced below, as shorthand.

Clause expanded extension $\otimes^{cl}_D$ is a unary operation that takes a context specifier clause c^{cl} and returns a set of worlds w.r.t. **D**. Clause expanded extension is performed with respect to a set of dimensions **D** and to the sets of dimension domains $\mathbf{V}_d$ for each d in **D**.

$\otimes^{cl}_D : C^{cl} \rightarrow S_W$, where C^{cl} is the set of all clauses, and S_W contains sets of worlds w.r.t. **D**

Clause expanded extension is defined below.

Definition 3.7

The ***clause expanded extension*** *of a context specifier clause* c^{cl}*, denoted* $\otimes^{cl}_D$ c^{cl}*, is defined as follows:* $\otimes^{cl}_D$ $c^{cl} = \times^{cl}$ $(\odot^{cl}_D$ $c^{cl})$.

Based on the above, Formula 3.1 can also take the form:

$W_D(c^{cl}) = \otimes^{cl}_D$ c^{cl}

3.3.1.2 Clause Equality

Clause equality $=^{cl}$ is a binary operation that takes two context specifier clauses c^{cl}_1, c^{cl}_2 and returns a Boolean value.

$=^{cl} : (C^{cl} \times C^{cl}) \rightarrow$ Boolean, where C^{cl} is the set of all clauses

Clause equality and inequality are defined as follows:

Definition 3.8

Let **D** *be a nonempty set of dimension names, and for each* $d \in$ **D** *let* $\mathbf{V}_d$ *be the domain of d, with* $\mathbf{V}_d \neq \varnothing$*. Let* c^{cl}_1, c^{cl}_2 *be context specifier clauses. Then* c^{cl}_1 *and* c^{cl}_2 *are* ***clause equal****, denoted* $c^{cl}_1 =^{cl} c^{cl}_2$*, iff* $W_D(c^{cl}_1) = W_D(c^{cl}_2)$*, where* $W_D(c^{cl})$ *is the set of worlds w.r.t.* **D** *represented by* c^{cl}*. Iff* $W_D(c^{cl}_1) \neq W_D(c^{cl}_2)$*, then* c^{cl}_1 *and* c^{cl}_2 *are* ***not clause equal****, denoted* $c^{cl}_1 \neq^{cl} c^{cl}_2$.

Although clause equality is defined by referring to some set of dimensions **D**, it is not actually defined with respect to any particular **D** [11]. The following Proposition 3.3 shows that if two context specifier clauses are clause equal w.r.t. a set of dimensions **D**, they are also clause equal w.r.t. any other **D′**. The symbol $=^{cl}_D$ is used in Proposition 3.3 to express clause equality w.r.t. **D**.

[11] However, checking two clauses for clause equality may need to take into account the domains of the dimensions encountered in those clauses.

Proposition 3.3

Let c^{cl}_1 and c^{cl}_2 be context specifier clauses and $\mathbf{D}$, $\mathbf{D'}$ *be sets of dimensions. The following hold: (a) if $c^{cl}_1 =^{cl}_D c^{cl}_2$ (with respect to* $\mathbf{D}$*), then $c^{cl}_1 =^{cl}_{D'} c^{cl}_2$ (with respect to* $\mathbf{D'}$*), and (b) if $c^{cl}_1 \neq^{cl}_D c^{cl}_2$ (with respect to* $\mathbf{D}$*), then $c^{cl}_1 \neq^{cl}_{D'} c^{cl}_2$ (with respect to* $\mathbf{D'}$*).*

Proof: If $\mathbf{D} = \mathbf{D'}$, claims (a) and (b) hold true. Lets assume that $\mathbf{D} \subset \mathbf{D'}$. If $c^{cl}_1 =^{cl}_D c^{cl}_2$ then $W_D(c^{cl}_1) = W_D(c^{cl}_2)$. Each world w_D in $W_D(c^{cl}_1)$, $W_D(c^{cl}_2)$ corresponds to a number of worlds w.r.t. $\mathbf{D'}$, as described in what follows. For every dimension d' such that $d' \in \mathbf{D'}$ and $d' \notin \mathbf{D}$, no constraint exists in contexts and $(d', \mathbf{V}_{d'})$ is implied for both c^{cl}_1 and c^{cl}_2, therefore a pair (d', v) must be added to $w_D \; \forall \; v \in \mathbf{V}_{d'}$ to get the set of worlds w.r.t. $\mathbf{D'}$ that correspond to w_D and are represented by c^{cl}_1 and c^{cl}_2. Therefore, $W_{D'}(c^{cl}_1) = W_{D'}(c^{cl}_2)$, and claim (a) holds when $\mathbf{D} \subset \mathbf{D'}$. Similarly, for claim (b) a different world w.r.t. $\mathbf{D}$ will lead to a (nonempty) set of different worlds w.r.t. $\mathbf{D'}$. It follows that if $c^{cl}_1 =^{cl}_{D'} c^{cl}_2$, then $c^{cl}_1 =^{cl}_D c^{cl}_2$, since as we have shown $c^{cl}_1 \neq^{cl}_D c^{cl}_2$ would imply that $c^{cl}_1 \neq^{cl}_{D'} c^{cl}_2$. If $\mathbf{D} \not\subset \mathbf{D'}$, we consider $\mathbf{D}_{min}$ for which $\mathbf{D}_{min} \subseteq \mathbf{D}$ and $\mathbf{D}_{min} \subseteq \mathbf{D'}$. We now have that $c^{cl}_1 =^{cl}_{Dmin} c^{cl}_2$ implies that $c^{cl}_1 =^{cl}_D c^{cl}_2$ and that $c^{cl}_1 =^{cl}_{D'} c^{cl}_2$. Therefore, claim (a) holds for any $\mathbf{D}$, $\mathbf{D'}$. Similarly, claim (b) holds for any $\mathbf{D}$, $\mathbf{D'}$.

Proposition 3.3 is based on the assumption, stated in Section 3.2.1, that the sets of dimensions we consider contain always (at least) those dimensions encountered in the contexts involved. Therefore, if $\mathbf{D}_{min}$ contains all the dimensions encountered in c^{cl}_1 and c^{cl}_2 and none other[12], $c^{cl}_1 =^{cl}_D c^{cl}_2 \Leftrightarrow c^{cl}_1 =^{cl}_{D'} c^{cl}_2$ and $c^{cl}_1 \neq^{cl}_D c^{cl}_2 \Leftrightarrow c^{cl}_1 \neq^{cl}_{D'} c^{cl}_2$, $\forall$ $\mathbf{D}$ and $\mathbf{D'}$ with $\mathbf{D}_{min} \subseteq \mathbf{D}$ and $\mathbf{D}_{min} \subseteq \mathbf{D'}$.

Conventional set-based equality is not sufficient for comparing directly two context specifier clauses, because it is possible for two clauses with different forms to represent the same set of worlds with respect to a set of dimensions $\mathbf{D}$. Two clauses are *clause equal* if they represent the same set of worlds w.r.t. some $\mathbf{D}$, regardless of the form they have.

Example 3.3

Let:

$\mathbf{D}$ = {lang, detail, format}
$\mathbf{V}_{lang}$ = {en, gr, sp}
$\mathbf{V}_{detail}$ = {low, high}
$\mathbf{V}_{format}$ = {ps, pdf}

c^{cl}_1 = {(lang,{en,gr}), (detail,{high}), (format,{ps,pdf})}
c^{cl}_2 = {(lang,{en,gr}), (detail,{high})}
c^{cl}_3 = {(lang,{en,gr}), (detail,{high}), (format,{})}
c^{cl}_4 = {(lang,{}), (detail,{low,high})}
c^{cl}_5 = {(detail,{low,high})}
c^{cl}_6 = {(format,{ps,pdf})}

Then, the following hold:

$c^{cl}_1 =^{cl} c^{cl}_2$
$c^{cl}_2 \neq^{cl} c^{cl}_3$

[12] In the trivial case where c^{cl}_1 and c^{cl}_2 do not contain any dimension specifiers (e.g. $c^{cl}_1 = c^{cl}_2 = \varnothing$), $\mathbf{D}_{min}$ cannot be defined since it must be a nonempty set. In this case, clause equality holds w.r.t. every possible set of dimensions.

$c^{cl}_3 =^{cl} c^{cl}_4$
$c^{cl}_5 =^{cl} c^{cl}_6$
♦

In Example 3.3, c^{cl}_1 and c^{cl}_2 represent the same set of worlds w.r.t. **D**, even though they have different forms. Also, according to Formula 3.1, c^{cl}_3 and c^{cl}_4 do not represent any world w.r.t. **D**, therefore they are clause equal. Clauses c^{cl}_5 and c^{cl}_6 are clause equal as well, since they represent every possible world w.r.t. **D**.

3.3.1.3 Empty Clause and Universal Clause

The **empty clause** $\mathcal{E}^{cl}$ is a common symbol for those clauses that do not represent any world w.r.t. some set of dimensions **D**. On the other hand, the **universal clause** $\mathcal{U}^{cl}$ is a common symbol for those clauses that do not effectively impose any constraints, and therefore imply every possible world w.r.t. some set of dimensions **D**.

Definition 3.9

Let $\mathbf{D}$ *be a nonempty set of dimension names, and for each* $d \in \mathbf{D}$ *let* $\mathbf{V}_d$ *be the domain of d, with* $\mathbf{V}_d \neq \emptyset$*. Let* c^{cl} *be a context specifier clause, and* $W_D(c^{cl})$ *be the set of worlds with respect to* $\mathbf{D}$ *represented by* c^{cl}*. Iff* $W_D(c^{cl}) = \emptyset$*, then* c^{cl} *is called* ***empty clause****, and is also denoted* $\mathcal{E}^{cl}$*. Iff* $W_D(c^{cl}) = U_D$*, where* U_D *is the universe of* $\mathbf{D}$*, then* c^{cl} *is called* ***universal clause****, and is also denoted* $\mathcal{U}^{cl}$*.*

Although the notions of empty clause and universal clause are defined by referring to some set of dimensions **D**, they are not actually defined with respect to any particular **D** [13]. In Section 3.2.1, we stated the requirement that a set of dimensions **D** must contain (at least) those dimensions encountered in the contexts involved. Let $\mathbf{D}_{min}$ contain all the dimensions encountered in a clause c^{cl} and none other. It is easy to see that if a clause c^{cl} is an empty clause w.r.t. **D**, then c^{cl} is also an empty clause w.r.t. any other $\mathbf{D}'$, where $\mathbf{D}_{min} \subseteq \mathbf{D}$ and $\mathbf{D}_{min} \subseteq \mathbf{D}'$. Similarly, if c^{cl} is a universal clause w.r.t. **D**, then c^{cl} is also a universal clause w.r.t. any other $\mathbf{D}'$, where $\mathbf{D}_{min} \subseteq \mathbf{D}$ and $\mathbf{D}_{min} \subseteq \mathbf{D}'$ [14].

Formula 3.1 together with Definition 3.9 gives:

$W_D(\mathcal{E}^{cl}) = \times^{cl} (\odot^{cl}_D \mathcal{E}^{cl}) = \emptyset$ **Formula 3.2**
$W_D(\mathcal{U}^{cl}) = \times^{cl} (\odot^{cl}_D \mathcal{U}^{cl}) = U_D$ **Formula 3.3**

From Definition 3.7, Formula 3.2 and Formula 3.3 can also take the form:

$W_D(\mathcal{E}^{cl}) = \otimes^{cl}_D \mathcal{E}^{cl} = \emptyset$
$W_D(\mathcal{U}^{cl}) = \otimes^{cl}_D \mathcal{U}^{cl} = U_D$

According to Formula 3.3 and the definitions of clause extension and clause expansion, if a clause c^{cl} contains a dimension specifier of the form (d,∅), then its expanded form will also contain (d,∅), and the extension of c^{cl} will give the empty set of worlds. In addition, for a clause to represent the empty set of worlds, it must contain at least one dimension specifier of

[13] However, determining whether a clause is an empty clause or a universal clause needs to take into account the domains of the dimensions encountered in that clause.
[14] In the trivial case where $c^{cl} = \emptyset$, $\mathbf{D}_{min}$ cannot be defined, since it must be a nonempty set. In this case, c^{cl} is a universal clause w.r.t. every possible set of dimensions.

the form (d,∅). Therefore, $\mathcal{E}^{cl}$ stands for every clause c^{cl}, such that (d,∅) ∈ c^{cl}. Those clauses are clause equal to one another since they all represent an empty set of worlds w.r.t. some **D**.

Similarly, if a clause c^{cl} contains exclusively dimension specifiers of the form (d,$\mathbf{V}_d$) or if c^{cl} = ∅, then its expanded form will contain exclusively dimension specifiers of the form (d,$\mathbf{V}_d$), and the extension of c^{cl} will give the universe of **D**, U_D. In addition, for a clause to represent U_D, it must contain exclusively dimension specifiers of the form (d,$\mathbf{V}_d$) or it must be the empty set. Therefore, $\mathcal{U}^{cl}$ stands for every clause c^{cl}, such that c^{cl} contains exclusively dimension specifiers of the form (d,$\mathbf{V}_d$), or c^{cl} = ∅. Those clauses are clause equal to one another since they all represent the universe U_D of some **D**.

Based on the above, it follows that $\mathcal{E}^{cl}$ actually stands for every member in a set of clause equal context specifier clauses, representing the empty set of worlds w.r.t. some **D**. Similarly, $\mathcal{U}^{cl}$ stands for every member in a set of clause equal context specifier clauses, representing the universe of some **D**.

Note that the expanded form w.r.t. **D** of an empty clause is again an empty clause, as is shown in Example 3.1 with c^{cl}_3 and $\odot^{cl}_D\ c^{cl}_3$. The case of c^{cl}_4 in Example 3.1 shows that the expanded form w.r.t. **D** of a universal clause is again a universal clause, as it covers the complete domains of all dimensions in **D**.

An interesting point is that clause ∅ is a universal clause for any possible set of dimensions. In what follows, we introduce a special symbol[15] representing a clause that is an empty clause for any possible set of dimensions.

Definition 3.10

*The symbol {–} stands for a clause containing dimension specifiers of the form (d,∅) ∀ d ∈ **D**.*

Thus, if **D** = {lang} then {–} = {(lang,∅)}, while if **D** = {detail, format} then {–} = {(detail,∅), (format,∅)}. Obviously, $\odot^{cl}_D$ {–} = {–} and W_D({–}) = ∅ for every possible set of dimensions.

In what follows, the sign of equality in expressions of the form $c^{cl} = \mathcal{E}^{cl}$ implies that c^{cl} is any of the clause equal context specifier clauses represented by $\mathcal{E}^{cl}$. The above equality can also be expressed using the clause equality $c^{cl} =^{cl}$ {–}. Similarly, the sign of equality in expressions of the form $c^{cl} = \mathcal{U}^{cl}$ implies that c^{cl} is any of the clause equal context specifier clauses represented by $\mathcal{U}^{cl}$. This equality can also be expressed using the clause equality $c^{cl} =^{cl}$ ∅.

3.3.1.4 Clause Intersection

Clause intersection $\cap^{cl}$ is a binary operation that takes two context specifier clauses c^{cl}_1, c^{cl}_2 and returns a context specifier clause c^{cl}_3.

$\cap^{cl} : (C^{cl} \times C^{cl}) \rightarrow C^{cl}$, where C^{cl} is the set of all clauses

Clause intersection is defined below.

---

[15] It is not obvious how to express a clause that is an empty clause w.r.t. every possible set of dimensions. To understand why, consider the expression {(lang,∅)}. This expression is an empty clause w.r.t. **D**={lang}, but is not even a clause w.r.t. **D′**={detail}, because, as stated in Section 3.2.1, all dimensions appearing in contexts must be included in the set of dimensions.

Definition 3.11

Let c^{cl}_1, c^{cl}_2 be two context specifier clauses. The ***clause intersection*** *$c^{cl}_1 \cap^{cl} c^{cl}_2$ is a context specifier clause $c^{cl}_3 = \{(d,V) \mid (d,V) \in c^{cl}_1$ and there is no pair (d,V') such that $(d,V') \in c^{cl}_2\} \cup \{(d,V) \mid (d,V) \in c^{cl}_2$ and there is no pair (d,V') such that $(d,V') \in c^{cl}_1\} \cup \{(d,V) \mid (d,V_1) \in c^{cl}_1$ and $(d,V_2) \in c^{cl}_2$ and $V = V_1 \cap V_2\}$.*

From Definition 3.11, if both c^{cl}_1, c^{cl}_2 do not contain any dimension specifiers, then the result will be the empty set as well (meaning every possible world): $\varnothing \cap^{cl} \varnothing = \varnothing$. In addition, $c^{cl} \cap^{cl} \{-\} = \{-\}$ for every set of dimensions.

Example 3.4

Consider the following:

$c^{cl}_1 = \{(lang,\{sp\}), (format,\{pdf\})\}$
$c^{cl}_2 = \{(lang,\{en,gr\}), (detail,\{med,high\})\}$
$c^{cl}_3 = \{(lang,\{gr,en,sp\})\}$
$c^{cl}_4 = \{(currency,\{\}), (lang,\{sp,gr\})\}$

$c^{cl}_1 \cap^{cl} c^{cl}_2 = \{(lang,\{\}), (format,\{pdf\}), (detail,\{med,high\})\}$
$c^{cl}_2 \cap^{cl} c^{cl}_3 = \{(lang,\{en,gr\}), (detail,\{med,high\})\}$
$c^{cl}_3 \cap^{cl} c^{cl}_4 = \{(currency,\{\}), (lang,\{sp,gr\})\}$
♦

It is easy to see from Definition 3.11, that for any context specifier clause c^{cl}, the clause intersection with the universal clause $\mathcal{U}^{cl}$ gives always a clause that is clause equal to c^{cl}, while the clause intersection with the empty clause $\mathcal{E}^{cl}$ gives always the empty clause.

$c^{cl} \cap^{cl} \mathcal{U}^{cl} =^{cl} c^{cl}$ **Formula 3.4**
$c^{cl} \cap^{cl} \mathcal{E}^{cl} = \mathcal{E}^{cl}$ **Formula 3.5**

Note that in Formula 3.4 clause equality is used instead of conventional equality. Therefore, the result of clause intersection in Formula 3.4 is not necessarily c^{cl}, but a context specifier clause that is clause equal to c^{cl}.

The clause intersection of two clauses represents the worlds that are common to both clauses, as is shown in the following proposition.

Proposition 3.4

If c^{cl}_1, c^{cl}_2 are context specifier clauses and $W_D(c^{cl}_1)$, $W_D(c^{cl}_2)$ are the sets of worlds w.r.t. ***D*** *that they represent, then $W_D(c^{cl}_1 \cap^{cl} c^{cl}_2) = W_D(c^{cl}_1) \cap W_D(c^{cl}_2)$.*

Proof: We will first show that:

$\odot^{cl}_D (c^{cl}_1 \cap^{cl} c^{cl}_2) = (\odot^{cl}_D c^{cl}_1) \cap^{cl} (\odot^{cl}_D c^{cl}_2)$

On the right side, the result of clause intersection will contain dimension specifiers that have a one-to-one correspondence with dimensions in **D**, because dimension specifiers in $\odot^{cl}_D c^{cl}_1$ and $\odot^{cl}_D c^{cl}_2$ have also a one-to-one correspondence with dimensions in **D**. This is also obvious for the left side of the expression. Therefore, for every (d,V_L) in the result of the left part there exists a (d,V_R) in the result of the right, and vice-versa. There are three cases for (d,V_L): (1) A corresponding dimension specifier (d,V) exists neither in c^{cl}_1 nor in c^{cl}_2. In this case, $(d,V_L) = (d,\mathbf{V}_d)$ as it has been added by the expansion in the left side. At the right side, both the expanded forms of c^{cl}_1 and c^{cl}_2 contain $(d,\mathbf{V}_d)$, thus the

result of clause intersection also contains $(d,\mathbf{V}_d)$. Therefore, $(d,V_L) = (d,V_R) = (d,\mathbf{V}_d)$. (2) A corresponding dimension specifier (d,V) exists in either c^{cl}_1 or c^{cl}_2, but not both. Then $(d,V_L) = (d,V)$, and $(d,V_R) = (d,V \cap \mathbf{V}_d) = (d,V)$. (3) Dimension specifiers (d,V) and (d,V′) exist in c^{cl}_1 and c^{cl}_2 respectively. In this case, the clause expansions do not affect the process and $(d,V_L) = (d,V_R) = (d,V \cap V')$. Thus, the above formula holds true. The next step is to show that:

$$(\times^{cl}\ (\odot^{cl}_D\ c^{cl}_1)) \cap (\times^{cl}\ (\odot^{cl}_D\ c^{cl}_2)) = \times^{cl}\ (\ (\odot^{cl}_D\ c^{cl}_1) \cap^{cl} (\odot^{cl}_D\ c^{cl}_2)\ )$$

Note that the left side contains a conventional set intersection, while the right side contains a clause intersection. In both the left and right sides, context specifier clauses participate in the expression in their expanded forms w.r.t. **D**. If either clause is the empty clause $\mathcal{E}^{cl}$, the result of both sides will be the empty set, and the equation will be true. Therefore, in what follows, we assume that neither clause contains a dimension specifier $(d,\varnothing)$. Suppose that one side still results into an empty set. If the result of the left side is an empty set, then there would be at least a dimension d for which $(d,V) \in (\odot^{cl}_D\ c^{cl}_1)$ and $(d,V') \in (\odot^{cl}_D\ c^{cl}_2)$ such that $V \cap V' = \varnothing$. If this was not the case and there was a common value v, since the extension takes all possible combinations of dimension values, there would exist a world in the result of the left side formed by such common values for each dimension. Since $V \cap V' = \varnothing$, then $(d, \varnothing) \in (\odot^{cl}_D\ c^{cl}_1) \cap^{cl} (\odot^{cl}_D\ c^{cl}_2)$ of the right side, and the result of the right side is also an empty set. In the same way it is easy to show that if the result of the right side is an empty set, the result of the left side is also an empty set. The only remaining case is for the result of both sides to be a nonempty set of worlds w.r.t. **D**. Each world will consist of pairs (d,v) that have a one-to-one correspondence with dimensions in **D**. In order for two worlds to be different, the one must contain a pair (d,v) and the other a pair (d,v′) with $v \neq v'$. Lets assume that there exists a world in the results of the left side that is not included in the results of the right side. Since clause extension transforms clauses to worlds by taking every possible combination of values in dimension specifiers, this world must contain at least one pair (d,v), such that *every* world in the results of the right side contains some pair (d,v′) with $v \neq v'$. However, if the result of the left side includes a world containing the pair (d,v), then the two expanded forms in the left side must contain dimension specifiers (d,V) and (d,V′) respectively, with $v \in (d,V)$ and $v \in (d,V')$. Therefore, $v \in (V \cap V')$ and there must be a world in the results of the right side containing the pair (d,v). So, every world in the results of the left side exists in the results of the right side as well. In the same way it is possible to show that every world in the results of the right side exists in the results of the left side. Thus, the above formula holds true.

Using Formula 3.1, and the two formulas above we have that: $W_D(c^{cl}_1) \cap W_D(c^{cl}_2) = (\times^{cl}\ (\odot^{cl}_D\ c^{cl}_1)) \cap (\times^{cl}\ (\odot^{cl}_D\ c^{cl}_2)) = \times^{cl}\ (\ (\odot^{cl}_D\ c^{cl}_1) \cap^{cl} (\odot^{cl}_D\ c^{cl}_2)\ ) = \times^{cl}\ (\odot^{cl}_D\ (c^{cl}_1 \cap^{cl} c^{cl}_2)) = W_D(c^{cl}_1 \cap^{cl} c^{cl}_2)$, which proves the initial claim.

Note that clause intersection is performed without taking into account any specific set of dimensions. In other words, if $c^{cl}_1 \cap^{cl} c^{cl}_2 = c^{cl}_3$ w.r.t. **D** and $c^{cl}_1 \cap^{cl} c^{cl}_2 = c^{cl}_4$ w.r.t. **D′**, then c^{cl}_3 and c^{cl}_4 are identical. In addition, as shown in Proposition 3.4, c^{cl}_3 represents the worlds w.r.t. **D** common to c^{cl}_1 and c^{cl}_2, and c^{cl}_4 represents the worlds w.r.t. **D′** common to c^{cl}_1 and c^{cl}_2.

Therefore, if $\mathbf{D}_{min}$ contains all the dimensions encountered in c^{cl}_1 and c^{cl}_2 and none other[16], the result of clause intersection can be correctly interpreted w.r.t. any **D**, with $\mathbf{D}_{min} \subseteq \mathbf{D}$.

Mutually Exclusive Clauses

A case of interest is when two clauses represent disjoint sets of worlds. Such clauses are called *mutually exclusive*.

Definition 3.12

*Two context specifier clauses c^{cl}_1, c^{cl}_2 are **mutually exclusive**, iff $W_D(c^{cl}_1) \cap W_D(c^{cl}_2) = \varnothing$, where $W_D(c^{cl})$ is the set of worlds represented by c^{cl} w.r.t. some set of dimensions **D**.*

From Proposition 3.4 we see that two clauses are mutually exclusive if and only if $c^{cl}_1 \cap^{cl} c^{cl}_2 = \mathcal{E}^{cl}$. Therefore, for two clauses to be mutually exclusive, the result of their clause intersection must be an empty clause.

Although mutually exclusiveness is defined by referring to some set of dimensions **D**, it is not actually defined with respect to any particular **D**. Since clause intersection does not depend on **D**, it follows that if two clauses are mutually exclusive w.r.t. some **D**, they are also mutually exclusive w.r.t. any other **D′**, where **D′** and **D** are supersets of $\mathbf{D}_{min}$.

It follows from Formula 3.4 that a universal clause $\mathcal{U}^{cl}$ is not mutually exclusive with any clause but the empty clause. From Formula 3.5 we see that an empty clause $\mathcal{E}^{cl}$ is mutually exclusive with any other clause, including empty clauses.

Example 3.5

Consider the following:

```
D = {lang, detail, format}
V_lang = {en, gr, sp}
V_detail = {low, high}
V_format = {ps, pdf}

c^cl_1 = {(lang,{sp}), (format,{pdf})}
c^cl_2 = {(lang,{en,gr}), (detail,{low,high})}
c^cl_3 = {(detail,{low,high})}
```

♦

In Example 3.5 c^{cl}_1 and c^{cl}_2 are mutually exclusive. In contrast, c^{cl}_1 and c^{cl}_3 are not mutually exclusive, since the worlds {(lang,sp), (format,pdf), (detail,low)} and {(lang,sp), (format,pdf), (detail,high)} are covered by both c^{cl}_1 and c^{cl}_3.

3.3.1.5 Clauses under Union and Difference

Context specifier clauses are *closed* under the operation of clause intersection, as it can be easily seen by its definition. However, it is not possible to define clause operations that would correspond to set union and set difference and that would exhibit the same property. What is more, there exist sets of worlds that cannot be expressed through a single context specifier clause; for instance, consider the set of worlds { {(lang,en),(detail,low)}, {(lang,gr),(detail,high)} }. Those problems lead to the introduction in Section 3.3.2 of **context specifiers**, which essentially are disjunctions of context specifier clauses. In

[16] In the trivial case where c^{cl}_1 and c^{cl}_2 do not contain any dimension specifiers, $\mathbf{D}_{min}$ cannot be defined since it must be a nonempty set. In this case, the result of clause intersection holds w.r.t. every possible set of dimensions.

sections 3.3.2.6 and 3.3.2.7, where union and difference for context specifiers are introduced, we also define union and difference for clauses. In those sections it will become clear that, in the general case, clause union and clause difference give context specifiers, and not context specifier clauses.

3.3.1.6 Clause Subset

Clause subset $\subseteq^{cl}$, and **proper clause subset** $\subset^{cl}$ are binary operations that take two context specifier clauses c^{cl}_1, c^{cl}_2 and return a Boolean value.

$\subseteq^{cl}$: (C^{cl} x C^{cl}) → Boolean, where C^{cl} is the set of all clauses
$\subset^{cl}$: (C^{cl} x C^{cl}) → Boolean, where C^{cl} is the set of all clauses

Clause subset / superset and proper clause subset / superset are defined as follows:

Definition 3.13

Let $\mathbf{D}$ *be a nonempty set of dimension names, and for each* $d \in \mathbf{D}$ *let* $\mathbf{V}_d$ *be the domain of d, with* $\mathbf{V}_d \neq \emptyset$. *Let* c^{cl}_1, c^{cl}_2 *be context specifier clauses. Then* c^{cl}_1 *is a* ***clause subset*** *of* c^{cl}_2, *denoted* $c^{cl}_1 \subseteq^{cl} c^{cl}_2$, *or equivalently,* c^{cl}_2 *is a* ***clause superset*** *of* c^{cl}_1, *denoted* $c^{cl}_2 \supseteq^{cl} c^{cl}_1$, *iff* $W_D(c^{cl}_1) \subseteq W_D(c^{cl}_2)$, *where* $W_D(c^{cl})$ *is the set of worlds w.r.t.* $\mathbf{D}$ *represented by* c^{cl}.

In case $W_D(c^{cl}_1) \subset W_D(c^{cl}_2)$, *then* c^{cl}_1 *is a* ***proper clause subset*** *of* c^{cl}_2, *denoted* $c^{cl}_1 \subset^{cl} c^{cl}_2$, *or equivalently,* c^{cl}_2 *is a* ***proper clause superset*** *of* c^{cl}_1, *denoted* $c^{cl}_2 \supset^{cl} c^{cl}_1$.

The following proposition shows that clause subset can be expressed in terms of clause intersection and clause equality, exactly like it happens with conventional set operations.

Proposition 3.5

For two context specifier clauses c^{cl}_1, c^{cl}_2 *the following hold: (a)* $c^{cl}_1 \subseteq^{cl} c^{cl}_2 \Leftrightarrow c^{cl}_1 \cap^{cl} c^{cl}_2 =^{cl} c^{cl}_1$, *and (b)* $c^{cl}_1 \subset^{cl} c^{cl}_2 \Leftrightarrow (c^{cl}_1 \subseteq^{cl} c^{cl}_2$ *AND* $c^{cl}_1 \neq^{cl} c^{cl}_2)$.

Proof: Given some set of dimensions $\mathbf{D}$, consider that the left side of expression (a) is true. Then, with respect to $\mathbf{D}$, $W_D(c^{cl}_1) \subseteq W_D(c^{cl}_2) \Leftrightarrow W_D(c^{cl}_1) \cap W_D(c^{cl}_2) = W_D(c^{cl}_1)$ and from Proposition 3.4 $W_D(c^{cl}_1 \cap^{cl} c^{cl}_2) = W_D(c^{cl}_1)$. From Definition 3.8 we have that $c^{cl}_1 \cap^{cl} c^{cl}_2 =^{cl} c^{cl}_1$. In the same way it is easy to show that, if the right side of expression (a) holds, then the left side holds as well. Expression (b) is obvious from the definitions of the operations involved.

Although clause (proper) subset / superset are defined by referring to some set of dimensions $\mathbf{D}$, they are not actually defined with respect to any particular $\mathbf{D}$ [17]. From Proposition 3.5, and because clause intersection and clause equality do not depend on $\mathbf{D}$, it is easy to see that, if $\mathbf{D}_{min}$ contains all the dimensions encountered in c^{cl}_1 and c^{cl}_2 and none other[18], then $c^{cl}_1 \subseteq^{cl}_D c^{cl}_2 \Leftrightarrow c^{cl}_1 \subseteq^{cl}_{D'} c^{cl}_2$ and $c^{cl}_1 \subset^{cl}_D c^{cl}_2 \Leftrightarrow c^{cl}_1 \subset^{cl}_{D'} c^{cl}_2$, $\forall$ $\mathbf{D}$ and $\mathbf{D'}$ with $\mathbf{D}_{min} \subseteq \mathbf{D}$ and $\mathbf{D}_{min} \subseteq \mathbf{D'}$.

[17] However, checking two clauses for clause (proper) subset / superset may need to take into account the domains of the dimensions encountered in those clauses.

[18] In the trivial case where c^{cl}_1 and c^{cl}_2 do not contain any dimension specifiers, $\mathbf{D}_{min}$ cannot be defined since it must be a nonempty set. In this case, the result of (proper) clause subset holds w.r.t. every possible set of dimensions.

Concerning the empty clause and the universal clause, it is obvious that $\mathcal{E}^{cl} \subseteq^{cl} c^{cl}$ and $c^{cl} \subseteq^{cl} \mathcal{U}^{cl}$, for any context specifier clause c^{cl}.

Example 3.6

Let:

$\mathbf{D} = \{lang, detail, format\}$
$\mathbf{V}_{lang} = \{en, gr, sp\}$
$\mathbf{V}_{detail} = \{low, high\}$
$\mathbf{V}_{format} = \{ps, pdf\}$

$c^{cl}_1 = \{(detail,\{low,high\})\}$
$c^{cl}_2 = \{(lang,\{en,gr\}), (detail,\{low,high\}), (format,\{ps\})\}$
$c^{cl}_3 = \{(lang,\{gr\}), (detail,\{high\}), (format,\{ps\})\}$
$c^{cl}_4 = \{(lang,\{gr,sp\}), (detail,\{low\})\}$

Then, the following hold:

$c^{cl}_3 \subset^{cl} c^{cl}_2 \subset^{cl} c^{cl}_1$
$c^{cl}_4 \subset^{cl} c^{cl}_1$
♦

3.3.2 Context Specifiers

In this section we introduce **context specifiers** (**contexts** for short), and define operations on context specifiers (**context operations**) based on the operations already defined for context specifier clauses (**clause operations**). As mentioned in Section 3.3.1.5, a context specifier clause cannot express any possible set of worlds. Context specifiers are structures able to represent any set of worlds.

Definition 3.14

*A **context specifier** c is a nonempty set of context specifier clauses, $c = \{c^{cl}_1, c^{cl}_2, ..., c^{cl}_n\}$ with $n \geq 1$.*

As already stated, the symbol ^{cl}, standing for *clause*, is used to identify clause variables and operations. We will not use any special symbol for context specifier variables. We will use the symbol ^{c}, standing for *context*, to disambiguate, whenever necessary, between context operations on the one hand, and clause operations and conventional set operations on the other.

3.3.2.1 Correspondence to Worlds

Essentially, a context specifier is a disjunction of context specifier clauses. The relation of context specifiers to worlds is explained in the following definition.

Definition 3.15

*The set of worlds w.r.t. **D** represented by a context specifier is given by the union of the sets of worlds w.r.t. **D** represented by the clauses contained in that context specifier.*

$W_D(c) = W_D(c^{cl}_1) \cup W_D(c^{cl}_2) \cup ... \cup W_D(c^{cl}_n)$, **Formula 3.6**
Where $c = \{c^{cl}_1, c^{cl}_2, ..., c^{cl}_n\}$, $n \geq 1$

The following proposition shows that any possible set of worlds can be expressed as a context specifier.

Proposition 3.6

*For any set of worlds $S = \{w_1, w_2, ..., w_n\}$ w.r.t. **D** there exists at least one context specifier c, such that $W_D(c) = S$.*

Proof: From Formula 3.2 and Formula 3.6, $W_D(\{\{(d,\varnothing)\}\}) = \varnothing$, thus if $S = \varnothing$ then $c = \{\{(d,\varnothing)\}\}$, for some d in **D**. If $S \neq \varnothing$, then the elements of S will be of the form $w_i = \{(d_1,v_1),(d_2,v_2),...,(d_k,v_k)\}$. Consider the clause $c^{cl}_i = \{(d_1,\{v_1\}),(d_2,\{v_2\}),...,(d_k,\{v_k\})\}$. It is obvious from Formula 3.1 that $W_D(c^{cl}_i) = \{w_i\}$. Now consider a context specifier c containing such clauses for every w_i in S. Then, $W_D(c) = S$.

Therefore, within the frame of a set of dimensions **D**, the terms *context*, *context specifier*, and *set of worlds* can be used interchangeably.

The proof of Proposition 3.6 shows that if **D** is finite and $\mathbf{V}_d$ is finite for every d in **D**, then there exists a finite context specifier representing any possible set of worlds. Note however, that a set of worlds can be represented by a number of different context specifiers. Since clauses in a context specifier may correspond to overlapping sets of worlds, a context specifier may contain an infinite number of clauses while still representing a finite set of worlds.

The set of worlds represented by a context specifier can be obtained from Formula 3.6, using the operations of clause expansion and clause extension defined in Section 3.3.1.1. Alternatively, it is possible to use directly the operations of context expansion and context extension, defined in what follows.

Context Expanded Form

Context expansion $\odot_D$ is a unary operation that takes a context specifier c_1 and returns a context specifier c_2. Context expansion is performed with respect to a set of dimensions **D** and to the sets of dimension domains $\mathbf{V}_d$ for each d in **D**.

$\odot_D$: C → C, where C is the set of all context specifiers

The resulting context c_2 is called the **expanded form** of context c_1 with respect to **D**, and is defined as follows:

Definition 3.16

*Let **D** be a set of dimension names, and $c = \{c^{cl}_1, c^{cl}_2, ..., c^{cl}_n\}$, $n \geq 1$, be a context specifier. Then the **context expansion** of c with respect to **D** (also called **expanded form** of c w.r.t. **D**), denoted $\odot_D$ c, is the context specifier $\{\odot^{cl}_D\ c^{cl}_1, \odot^{cl}_D\ c^{cl}_2, ..., \odot^{cl}_D\ c^{cl}_n\}$.*

Clauses of the expanded form of a context specifier contain dimension specifiers that correspond exactly to dimensions in **D**. The expanded form of a context specifier is equivalent to that context specifier with respect to **D**, in the sense that they both represent the same set of worlds with respect to **D**:

Proposition 3.7

*A context specifier and its expanded form with respect to **D** represent the same set of worlds w.r.t. **D**, for any set of dimensions **D**.*

Proof: The claim is obvious from Formula 3.6 and Proposition 3.1.

Notice that *the expanded form of a context can never contain the empty set*, since clause expansion does never return the empty set.

Context Extension

Context extension $\times$ is a unary operation that takes a context specifier c, with $\varnothing \notin c$, and returns a set of sets of *(dimension, value)* pairs.

$\times : (C\text{-}C_{\varnothing}) \rightarrow S$, where C is the set of all context specifiers, and $C_{\varnothing}$ is the set of contexts $c_{\varnothing}$ with $\varnothing \in c_{\varnothing}$

Context extension is defined below.

Definition 3.17

*The **context extension** of a context specifier $c = \{c^{cl}_1, c^{cl}_2, ..., c^{cl}_n\}$, $n \geq 1$, is denoted $\times c$, and is the set $\{\times^{cl} c^{cl}_1\} \cup \{\times^{cl} c^{cl}_2\} \cup ... \cup \{\times^{cl} c^{cl}_n\}$.*

Each set in the result of a *clause* extension contains *(dimension, value)* pairs for exactly the same dimensions. In contrast to clause extension, the result of context extension may contain sets of different cardinality consisting of pairs that correspond to different dimensions. However, the context extension of the context expansion w.r.t. **D** of a context specifier will give a set of worlds w.r.t. **D**, exactly as it happens with clauses.

Proposition 3.8

*The worlds w.r.t. **D** represented by a context specifier c are given by the context extension of the expanded form w.r.t. **D** of c.*

$W_D(c) = \times (\odot_D\ c)$ **Formula 3.7**

Proof: Let $c = \{c^{cl}_1, c^{cl}_2, ..., c^{cl}_n\}$, $n \geq 1$, be a context specifier. Then, according to Definition 3.16 and Definition 3.17, $\odot_D\ c = \{\odot^{cl}_D\ c^{cl}_1, \odot^{cl}_D\ c^{cl}_2, ..., \odot^{cl}_D\ c^{cl}_n\}$, and $\times (\odot_D\ c) = \{\times^{cl} (\odot^{cl}_D\ c^{cl}_1)\} \cup \{\times^{cl} (\odot^{cl}_D\ c^{cl}_2)\} \cup ... \cup \{\times^{cl} (\odot^{cl}_D\ c^{cl}_n)\}$. The last expression, according to Formula 3.1, is equal to $W_D(c^{cl}_1) \cup W_D(c^{cl}_2) \cup ... \cup W_D(c^{cl}_n)$, which according to Formula 3.6 is equal to $W_D(c)$.

Context Expanded Extension

Context expansion and context extension are often used together to give the set of worlds w.r.t. **D** represented by a context specifier. An operation that combines context expansion and context extension is introduced below, as shorthand.

Context expanded extension $\otimes_D$ is a unary operation that takes a context specifier c and returns a set of worlds w.r.t. **D**. Context expanded extension is performed with respect to a set of dimensions **D** and to the sets of dimension domains $\mathbf{V}_d$ for each d in **D**.

$\otimes_D : C \rightarrow S_W$, where C is the set of all context specifiers, and S_W contains sets of worlds w.r.t. **D**

Context expanded extension is defined below.

Definition 3.18

*The **context expanded extension** with respect to **D** of a context specifier c, denoted $\otimes_D\ c$, is defined as follows: $\otimes_D\ c = \times (\odot_D\ c)$.*

Based on the above, Formula 3.7 can also take the form:

$W_D(c) = \otimes_D c$

3.3.2.2 Context Equality

Context equality $=^c$ is a binary operation that takes two context specifiers c_1, c_2 and returns a Boolean value.

$=^c : (C \times C) \rightarrow$ Boolean, where C is the set of all contexts

Context equality and inequality are defined as follows:

Definition 3.19

Let **D** *be a nonempty set of dimension names, and* c_1, c_2 *be context specifiers. Then* c_1 *and* c_2 *are* ***context equal****, denoted by* $c_1 =^c c_2$*, iff* $W_D(c_1) = W_D(c_2)$*, where* $W_D(c)$ *is the set of worlds w.r.t.* **D** *represented by c. Iff* $W_D(c_1) \neq W_D(c_2)$*, then* c_1 *and* c_2 *are* ***not context equal****, denoted by* $c_1 \neq^c c_2$.

Although context equality is defined by referring to some set of dimensions **D**, it is not actually defined with respect to any particular **D** [19]. The following Proposition 3.9 shows that if two context specifiers are context equal w.r.t. a set of dimensions **D**, they are also context equal w.r.t. any other **D′**. The symbol $=^c_D$ is used in Proposition 3.9 to express context equality w.r.t. **D**.

Proposition 3.9

Let c_1 *and* c_2 *be context specifiers and* **D**, **D′** *be sets of dimensions. The following hold: (a) if* $c_1 =^c_D c_2$ *(with respect to* **D***), then* $c_1 =^c_{D'} c_2$ *(with respect to* **D′***), and (b) if* $c_1 \neq^c_D c_2$ *(with respect to* **D***), then* $c_1 \neq^c_{D'} c_2$ *(with respect to* **D′***).*

Proof: Similar to that of Proposition 3.3.

Proposition 3.9 is based on the assumption, stated in Section 3.2.1, that the sets of dimensions we consider contain always (at least) those dimensions encountered in the contexts involved. Therefore, if $\mathbf{D}_{min}$ contains all the dimensions encountered in c_1 and c_2 and none other, $c_1 =^c_D c_2 \Leftrightarrow c_1 =^c_{D'} c_2$ and $c_1 \neq^c_D c_2 \Leftrightarrow c_1 \neq^c_{D'} c_2$, $\forall$ **D** and **D′** with $\mathbf{D}_{min} \subseteq \mathbf{D}$ and $\mathbf{D}_{min} \subseteq \mathbf{D'}$.

Conventional set-based equality is not sufficient for comparing context specifiers directly, because it is possible for two context specifiers with different form to represent the same set of worlds w.r.t. **D**. Two context specifiers are *context equal* if they represent the same set of worlds w.r.t. some **D**, regardless of the form they have.

Example 3.7

Let:

$\mathbf{D}$ = {lang, detail, format}
$\mathbf{V}_{lang}$ = {en, gr, sp}
$\mathbf{V}_{detail}$ = {low, high}
$\mathbf{V}_{format}$ = {ps, pdf}

[19] However, checking two context specifiers for context equality may need to take into account the domains of the dimensions encountered in those context specifiers.

```
c1 = { {(lang,{en,gr}),(detail,{high})},
       {(lang,{gr}),(detail,{low})} }
c2 = { {(lang,{en}),(detail,{high})},
       {(lang,{gr}),(detail,{low,high})} }
c3 = { {(detail,{low,high}),(format,{ps,pdf})} }
c4 = { {(lang,{gr,sp}),(detail,{low,high})},
       {(lang,{en}),(detail,{low,high})} }
```

Then, the following hold:

$c_1 =^c c_2$
$c_2 \neq^c c_3$
$c_3 =^c c_4$
♦

In Example 3.7, contexts c_1 and c_2 have different forms but represent the same set of worlds w.r.t. **D**. Context c_3 contains a universal clause $\mathcal{U}^{cl}$ that represents every possible world w.r.t. **D**, and is not context equal to c_1 and c_2. The same is the case of c_4, which contains two clauses that together cover completely U_D.

3.3.2.3 Empty Context and Universal Context

Like their clause counterparts, the **empty context** $\mathcal{E}$ is a common symbol for those contexts that do not represent any world w.r.t. some set of dimensions **D**. The **universal context** $\mathcal{U}$ is a common symbol for those contexts that represent the universe U_D of some **D**.

Definition 3.20

Let $\mathbf{D}$ *be a nonempty set of dimension names, and for each* $d \in \mathbf{D}$ *let* $\mathbf{V}_d$ *be the domain of d, with* $\mathbf{V}_d \neq \emptyset$*. Let c be a context specifier, and* $W_D(c)$ *the set of worlds it represents with respect to* $\mathbf{D}$*. Iff* $W_D(c) = \emptyset$*, then c is called* ***empty context****, and is also denoted* $\mathcal{E}$*. Iff* $W_D(c) = U_D$*, where* U_D *is the universe of* $\mathbf{D}$*, then c is called* ***universal context****, and is also denoted* $\mathcal{U}$*.*

Although empty context and universal context are defined by referring to some set of dimensions **D**, they are not actually defined with respect to any particular **D** [20]. In Section 3.2.1, we stated the requirement that a set of dimensions **D** must contain (at least) the dimensions encountered in the contexts involved. Let $\mathbf{D}_{min}$ contain all the dimensions encountered in a context specifier c and none other. It is easy to see that if c is an empty context w.r.t. **D**, then c is also an empty context w.r.t. any other $\mathbf{D}'$, where $\mathbf{D}_{min} \subseteq \mathbf{D}$ and $\mathbf{D}_{min} \subseteq \mathbf{D}'$. Similarly, if c is a universal context w.r.t. **D**, then c is also a universal context w.r.t. any other $\mathbf{D}'$, where $\mathbf{D}_{min} \subseteq \mathbf{D}$ and $\mathbf{D}_{min} \subseteq \mathbf{D}'$.

Formula 3.7 together with Definition 3.20 gives:

$W_D(\mathcal{E}) = \times (\odot_D \mathcal{E}) = \emptyset$ **Formula 3.8**
$W_D(\mathcal{U}) = \times (\odot_D \mathcal{U}) = U_D$ **Formula 3.9**

From Definition 3.18, Formula 3.8 and Formula 3.9 can also take the form:

$W_D(\mathcal{E}) = \otimes_D \mathcal{E} = \emptyset$

[20] However, determining whether a context specifier is an empty context or a universal context needs to take into account the domains of the dimensions encountered in that context specifier.

$W_D(\mathcal{U}) = \otimes_D \mathcal{U} = U_D$

From Formula 3.6 we see that an empty context can contain only empty clauses. Thus $\mathcal{E}$ stands for every context specifier of the form $\{\mathcal{E}^{cl}, ..., \mathcal{E}^{cl}, ...\}$. All those contexts are context equal to one another, since they represent the empty set of worlds w.r.t. some **D**.

Likewise, every context specifier of the form $\{c^{cl}_1,..., \mathcal{U}^{cl},...,c^{cl}_m\}$ that contains at least one universal clause, is represented by the universal context $\mathcal{U}$. Apart context specifiers containing the universal clause, the universal context $\mathcal{U}$ covers another kind of context specifiers, as is shown in the case of c_3 in Example 3.8.

Example 3.8

Let:

$\mathbf{D}$ = {lang, detail}
$\mathbf{V}_{lang}$ = {en, gr, sp}
$\mathbf{V}_{detail}$ = {low, high}

c_1 = { {(detail,{low,high})} }
c_2 = { {(detail,{low}),(lang,{en,sp})}, {} }
c_3 = { {(lang,{gr,sp}),(detail,{low,high})},
{(lang,{en}),(detail,{low,high})} }

♦

In Example 3.8, context specifiers c_1, c_2, and c_3 are universal contexts. Each of contexts c_1 and c_2 contains a universal clause. Context c_3 on the other hand, does not contain any universal clause. However, the two clauses of c_3 taken together represent all possible worlds w.r.t. **D**. All contexts represented by $\mathcal{U}$ are context equal to one another, since they represent the universe U_D of a set of dimensions **D**.

From the above, it is evident that $\mathcal{E}$ actually stands for every member in a set of context equal context specifiers, representing the empty set of worlds w.r.t. some **D**. Similarly, $\mathcal{U}$ stands for every member in a set of context equal context specifiers, representing the universe of some **D**.

Note that the expanded form w.r.t. **D** of an empty context $\mathcal{E}$ is again an empty context, while the expanded form w.r.t. **D** of a universal context $\mathcal{U}$ is again a universal context.

An interesting point is that the context $\{\varnothing\}$ is a universal context with respect to every possible set of dimensions, and the context $\{\{-\}\}$ is an empty context with respect to every possible set of dimensions.

In what follows, the sign of equality in expressions of the form $c = \mathcal{E}$ implies that c is any of the context equal context specifiers represented by $\mathcal{E}$. The above equality can also be expressed using the context equality $c =^c \{\{-\}\}$. Similarly, the sign of equality in expressions of the form $c = \mathcal{U}$ implies that c is any of the context equal context specifiers represented by $\mathcal{U}$. This equality can also be expressed using the context equality $c =^c \{\varnothing\}$.

3.3.2.4 Simplification of Context Specifiers

As we have shown in Proposition 3.6, context specifiers can represent any set of worlds by encompassing a number of context specifier clauses. It is important however to realize that clauses in a context specifier may represent overlapping sets of worlds, sets that are subsets of others, etc. Although this redundancy does not affect the operations on context specifiers

per se, it can mean increased complexity for context operations. It is therefore useful to express context specifiers in an economical way by eliminating redundant elements.

Example 3.9

This example demonstrates the possible simplification cases. It starts with a number of clauses and continues with combinations of those clauses in context specifiers.

c^{cl}_1 = {(lang,{en,gr})}
c^{cl}_2 = {(lang,{en,gr}), (detail,{low,high})}
c^{cl}_3 = {(lang,{en,gr}), (detail,{low})}
c^{cl}_4 = {(lang,{sp,gr}), (detail,{low})}

$c_1 = \{c^{cl}_a, \mathcal{U}^{cl}, c^{cl}_b, \ldots\} =^c \{\mathcal{U}^{cl}\} = \mathcal{U}$
$c_2 = \{c^{cl}_a, \mathcal{E}^{cl}, c^{cl}_b, \ldots\} =^c \{c^{cl}_a, c^{cl}_b, \ldots\}$
$c_3 = \{\mathcal{E}^{cl}, \mathcal{E}^{cl}, \ldots, \mathcal{E}^{cl}\} =^c \{\mathcal{E}^{cl}\} = \mathcal{E}$
$c_4 = \{c^{cl}_1, c^{cl}_2\} =^c \{c^{cl}_1\}$
$c_5 = \{c^{cl}_2, c^{cl}_3\} =^c \{c^{cl}_2\}$
$c_6 = \{c^{cl}_3, c^{cl}_4\} =^c \{ \{(lang,\{en,gr,sp\}), (detail,\{low\})\} \}$
$c_7 = \{c^{cl}_2, c^{cl}_3, c^{cl}_4\} =^c \{c^{cl}_2, c^{cl}_4\} =^c$
$\{ c^{cl}_2, \{(lang,\{en,gr,sp\}), (detail,\{low\})\} \}$

♦

Note that in Example 3.9, $c^{cl}_3 \subseteq^{cl} c^{cl}_2 \subseteq^{cl} c^{cl}_1$. This explains the possible alternative forms of contexts c_4 and c_5, since clauses that are clause subsets do not contribute any new worlds to a context. The same holds true for c_1, c_2 and c_3, since $\mathcal{E}^{cl} \subseteq^{cl} \mathcal{E}^{cl} \subseteq^{cl} c^{cl} \subseteq^{cl} \mathcal{U}^{cl}$ for any clause c^{cl}. Thus, a possible simplification of a context specifier involves the elimination of clauses that are subsets of other clauses in that context specifier.

If a context specifier contains a universal clause $\mathcal{U}^{cl}$, all the other clauses in the context specifier can be omitted, since the universal clause represents already every possible world. This is the case of c_1 in Example 3.9.

In a context specifier, empty clauses $\mathcal{E}^{cl}$ do not contribute any worlds to the final result. Therefore, if such clauses are omitted from a context specifier, the set of worlds represented by that context specifier will not change. Note, however, that it is not possible to omit a clause if it is the only one in a context specifier, because a context specifier is defined as a nonempty set. Those cases are shown by c_2 and c_3 in Example 3.9.

Context c_6 shows that two or more clauses can be combined in a single clause. This is possible only if the clauses differ in just one dimension specifier concerning the same dimension. Then the resulting clause would consist of all other dimension specifiers (which must be common to both clauses) plus the dimension specifier (d, $V \cup V'$), where (d,V) belongs to the first clause and (d,V′) belongs to the second. This can be considered as a case of clause union returning one clause, as explained in section 3.3.2.6, which nevertheless is not always possible as demonstrated in Example 3.11.

The process of simplification does not always lead to a context specifier of the same form. In other worlds, if two contexts are context equal, their simplified forms may not be equal in the conventional set-based sense (although they will continue to be context equal). This is shown in Example 3.9 by the context specifier c_7, which can be transformed into two context specifiers with different form, for which no more simplification is possible.

Dimension domains must be taken into account for determining universal contexts, and for dealing with dimension specifiers of the form $(d,\mathbf{V}_d)$ which may lead to further simplification of clauses.

3.3.2.5 Context Intersection

Context intersection $\cap^c$ is a binary operation that takes two context specifiers c_1, c_2 and returns a context specifier c_3.

$\cap^c$: (C x C) → C, where C is the set of all context specifiers

Context intersection is defined as follows:

Definition 3.21

Let c_1, c_2 be two context specifiers. The ***context intersection*** $c_1 \cap^c c_2$ *is a context specifier* $c_3 = \{c^{cl}_n \cap^{cl} c^{cl}_m \mid c^{cl}_n \in c_1, c^{cl}_m \in c_2\}$.

Context intersection involves the clause intersection of every pair of clauses from the context specifiers.

Example 3.10
Consider the following:

c_1 = { {(lang,{en,gr}),(detail,{high})} }
c_2 = { {(lang,{en}),(detail,{low})}, {(lang,{gr})} }
c_3 = { $\mathcal{E}^{cl}$, {(format,{ps,pdf})} }

$c_1 \cap^c c_2$ =
{ $\mathcal{E}^{cl}$, {(lang,{gr}),(detail,{high})} }
$c_1 \cap^c c_3$ =
{ $\mathcal{E}^{cl}$, {(lang,{en,gr}),(detail,{high}),(format,{ps,pdf})} }
♦

The context intersection of two context specifiers represents the worlds that are common to both context specifiers, as is shown in the following proposition.

Proposition 3.10

If c_1, c_2 are context specifiers and $W_D(c_1)$, $W_D(c_2)$ are the sets of worlds w.r.t. ***D*** *that they represent, then $W_D(c_1 \cap^c c_2) = W_D(c_1) \cap W_D(c_2)$.*

Proof: Let $c_1 = \{c^{cl}_{11}, c^{cl}_{12},..., c^{cl}_{1n}\}$ and $c_2 = \{c^{cl}_{21}, c^{cl}_{22},..., c^{cl}_{2m}\}$ with n,m ≥ 1. Then, using in sequence Definition 3.21, Formula 3.6, Proposition 3.4, the distributive property of set union and intersection, and Formula 3.6 again we have that: $W_D(c_1 \cap^c c_2) = W_D(\{c^{cl}_{11} \cap^{cl} c^{cl}_{21}, c^{cl}_{11} \cap^{cl} c^{cl}_{22},..., c^{cl}_{1n} \cap^{cl} c^{cl}_{2m}\}) = W_D(c^{cl}_{11} \cap^{cl} c^{cl}_{21}) \cup W_D(c^{cl}_{11} \cap^{cl} c^{cl}_{22}) \cup ... \cup W_D(c^{cl}_{1n} \cap^{cl} c^{cl}_{2m}) = [W_D(c^{cl}_{11}) \cap W_D(c^{cl}_{21})] \cup [W_D(c^{cl}_{11}) \cap W_D(c^{cl}_{22})] \cup ... \cup [W_D(c^{cl}_{1n}) \cap W_D(c^{cl}_{2m})] = [W_D(c^{cl}_{11}) \cup W_D(c^{cl}_{12}) \cup ... \cup W_D(c^{cl}_{1n})] \cap [W_D(c^{cl}_{21}) \cup W_D(c^{cl}_{22}) \cup ... \cup W_D(c^{cl}_{2m})] = W_D(c_1) \cap W_D(c_2)$.

From Proposition 3.10 it follows that the context intersection of any context specifier with the empty context $\mathcal{E}$ gives always the empty context $\mathcal{E}$. In addition, for any context specifier c, the context intersection with the universal context $\mathcal{U}$ gives always a context specifier that represents the same worlds as c, w.r.t. any set of dimensions **D**.

$c \cap^c \mathcal{E} = \mathcal{E}$ **Formula 3.10**
$c \cap^c \mathcal{U} =^c c$ **Formula 3.11**

Note that in Formula 3.11 context equality is used instead of conventional equality. Therefore, the result of context intersection in Formula 3.11 is not necessarily c, but a context specifier that is context equal to c.

Like clause intersection, *context intersection is performed without taking into account any specific set of dimensions*. In other words, if $c_1 \cap^c c_2 = c_3$ w.r.t. **D** and $c_1 \cap^c c_2 = c_4$ w.r.t. **D′**, then c_3 and c_4 are identical. In addition, as shown in Proposition 3.10, c_3 represents the worlds w.r.t. **D** common to c_1 and c_2, and c_4 represents the worlds w.r.t. **D′** common to c_1 and c_2. Therefore, if $\mathbf{D}_{min}$ contains all the dimensions encountered in c_1 and c_2 and none other, the result of a context intersection can be correctly interpreted w.r.t. any **D**, with $\mathbf{D}_{min} \subseteq \mathbf{D}$.

Mutually Exclusive Contexts

Two context specifiers are mutually exclusive if they represent disjoint sets of worlds.

Definition 3.22

*Two context specifiers c_1, c_2 are **mutually exclusive**, iff $W_D(c_1) \cap W_D(c_2) = \varnothing$, where $W_D(c)$ is the set of worlds represented by c w.r.t. some set of dimensions **D**.*

From Proposition 3.10 we see that two contexts are mutually exclusive if and only if $c_1 \cap^c c_2 = \mathcal{E}$. Therefore, for two contexts to be mutually exclusive, the result of their context intersection must be an empty context $\mathcal{E}$.

Although context mutually exclusiveness is defined by referring to some set of dimensions **D**, it is not actually defined with respect to any particular **D**. Since context intersection does not depend on **D**, it follows that if two contexts are mutually exclusive w.r.t. **D**, they are also mutually exclusive w.r.t. any **D′**, where **D′** and **D** are supersets of $\mathbf{D}_{min}$.

It follows from Formula 3.10 that an empty context $\mathcal{E}$ is mutually exclusive with any other context, including empty contexts. From Formula 3.11 we see that a universal context $\mathcal{U}$ is not mutually exclusive with any context but the empty context.

3.3.2.6 Context Union and Clause Union

In section 3.3.1.5 we mentioned that context specifier clauses are not closed under the operations of union and difference. Although in some cases the union of two clauses can be represented as a single clause, in the general case it can only be represented as a context specifier. Context specifiers, on the other hand, *are closed* under the operation of context union. Before defining context union, we will define clause union. Clause union is not used in the definition of context union and is included for completeness.

Clause Union

Clause union $\cup^{cl}$ is a binary operation that takes two context specifier clauses c^{cl}_1, c^{cl}_2 and returns a context specifier c.

$\cup^{cl} : (C^{cl} \times C^{cl}) \rightarrow C$, where C^{cl} is the set of all clauses, and C is the set of all context specifiers

Clause union is defined as follows:

Definition 3.23

*Let c^{cl}_1, c^{cl}_2 be two context specifier clauses. The **clause union** $c^{cl}_1 \cup^{cl} c^{cl}_2$ is the context specifier $\{c^{cl}_1, c^{cl}_2\}$.*

Example 3.11
Consider the following:

c^{cl}_1 = {(lang,{en,gr}),(detail,{high})}
c^{cl}_2 = {(lang,{en,gr}),(detail,{low})}
c^{cl}_3 = {(lang,{sp}),(detail,{low})}

$c^{cl}_1 \cup^{cl} c^{cl}_2$ =
{ {(lang,{en,gr}),(detail,{high})},
{(lang,{en,gr}),(detail,{low})} }
$=^c$ { {(lang,{en,gr}),(detail,{high,low})} }
$c^{cl}_2 \cup^{cl} c^{cl}_3$ =
{ {(lang,{en,gr}),(detail,{low})},
{(lang,{sp}),(detail,{low})} }
$=^c$ { {(lang,{en,gr,sp}),(detail,{low})} }
$c^{cl}_1 \cup^{cl} c^{cl}_3$ =
{ {(lang,{en,gr}),(detail,{high})},
{(lang,{sp}),(detail,{low})} }
♦

It is interesting to note that in the case of $c^{cl}_1 \cup^{cl} c^{cl}_2$ and $c^{cl}_2 \cup^{cl} c^{cl}_3$ the result is context equal to a context containing only one clause; therefore in those cases the union of two clauses can be expressed more compactly as a single clause. In the case of $c^{cl}_1 \cup^{cl} c^{cl}_3$ however, such a "factorization" is not possible, and the result can only be expressed as a context specifier.

Proposition 3.11

If c^{cl}_1, c^{cl}_2 are context specifier clauses and $W_D(c^{cl}_1)$, $W_D(c^{cl}_2)$ are the sets of worlds w.r.t. some $\mathbf{D}$ *that they represent, then $W_D(c^{cl}_1 \cup^{cl} c^{cl}_2) = W_D(c^{cl}_1) \cup W_D(c^{cl}_2)$.*

Proof: The claim is obvious from Definition 3.23 and Formula 3.6.

Note that *clause union is performed without taking into account any specific set of dimensions*. Therefore, if $\mathbf{D}_{min}$ contains all the dimensions encountered in c_1 and c_2 and none other, the result of a clause union can be correctly interpreted w.r.t. any $\mathbf{D}$, with $\mathbf{D}_{min} \subseteq \mathbf{D}$.

Context Union

Context union $\cup^c$ is a binary operation that takes two context specifiers c_1, c_2 and returns a context specifier c_3.

$\cup^c$: (C x C) → C, where C is the set of all context specifiers

Context union is defined as follows:

Definition 3.24

Let c_1, c_2 be two context specifiers. The ***context union*** *$c_1 \cup^c c_2$ is the context specifier $c_1 \cup c_2$.*

Therefore, the context union of two context specifiers contains the clauses of both context specifiers. Context union is nothing else than conventional set union, and is introduced for reasons of uniformity of notation.

Example 3.12
Consider the following:

```
c1 = { {(lang,{en,gr}),(detail,{high})} }
c2 = { {(lang,{en}),(detail,{low})}, {(lang,{gr})} }

c1 ∪c c2 = c1 ∪ c2 =
  { {(lang,{en,gr}),(detail,{high})},
    {(lang,{en}),(detail,{low})}, {(lang,{gr})} }
```
♦

The context union of two context specifiers represents the union of the worlds that are specified by the two contexts, as is shown in the following proposition.

Proposition 3.12

If c_1, c_2 are context specifiers and $W_D(c_1)$, $W_D(c_2)$ are the sets of worlds w.r.t. **D** *that they represent, then $W_D(c_1 \cup^c c_2) = W_D(c_1) \cup W_D(c_2)$.*

Proof: Let $c_1 = \{c^{cl}_{11}, c^{cl}_{12},..., c^{cl}_{1n}\}$ and $c_2 = \{c^{cl}_{21}, c^{cl}_{22},..., c^{cl}_{2m}\}$ with n,m ≥ 1. Then, using Definition 3.24 and Formula 3.6, we have that: $W_D(c_1 \cup^c c_2) = W_D(\{c^{cl}_{11}, c^{cl}_{12},..., c^{cl}_{1n}, c^{cl}_{21}, c^{cl}_{22},..., c^{cl}_{2m}\}) = W_D(c^{cl}_{11}) \cup W_D(c^{cl}_{12}) \cup ... \cup W_D(c^{cl}_{1n}) \cup W_D(c^{cl}_{21}) \cup W_D(c^{cl}_{22}) \cup ... \cup W_D(c^{cl}_{2m}) = W_D(c_1) \cup W_D(c_2)$.

From Proposition 3.12, the context union of any context specifier with a universal context $\mathcal{U}$ gives always a universal context $\mathcal{U}$. For any context specifier c, the context union with an empty context $\mathcal{E}$ gives always a context specifier that represents the same worlds as c.

$c \cup^c \mathcal{U} = \mathcal{U}$ **Formula 3.12**

$c \cup^c \mathcal{E} =^c c$ **Formula 3.13**

Note that in Formula 3.13 context equality is used instead of conventional equality. Therefore, the result of context union in Formula 3.13 is not necessarily c, but a context specifier that is context equal to c.

Like context intersection, *context union is performed without taking into account any specific set of dimensions*. In other words, if $c_1 \cup^c c_2 = c_3$ w.r.t. **D** and $c_1 \cup^c c_2 = c_4$ w.r.t. **D′**, then c_3 and c_4 are identical. In addition, as shown in Proposition 3.12, c_3 represents the worlds w.r.t. **D** of both c_1 and c_2, and c_4 represents the worlds w.r.t. **D′** of both c_1 and c_2. Therefore, if $\mathbf{D}_{min}$ contains all the dimensions encountered in c_1 and c_2 and none other, the result of a context union can be correctly interpreted w.r.t. any **D**, with $\mathbf{D}_{min} \subseteq \mathbf{D}$.

3.3.2.7 Context Difference and Clause Difference

In section 3.3.1.5 we mentioned that context specifier clauses are *not closed* under the operation of difference. Just like we have seen in clause union, there are cases where the difference of two clauses can be represented as a single clause, in the general case however it can only be represented as a context specifier. Context specifiers are closed under the operation of context difference. Before defining context difference, we will discuss clause difference.

Clause Difference

Clause difference $-^{cl}$ is a binary operation that takes two context specifier clauses c^{cl}_1, c^{cl}_2 and returns a context specifier c.

```
−cl : (Ccl x Ccl) → C, where Ccl is the set of all clauses, and
                       C is the set of context specifiers
```

Clause difference is defined below.

Definition 3.25

Let c^{cl}_1, c^{cl}_2 be two context specifier clauses. The ***clause difference*** *of c^{cl}_1 and c^{cl}_2, denoted $c^{cl}_1 -^{cl} c^{cl}_2$, is a context specifier c defined as follows:*

1. *If $(d,\emptyset) \in c^{cl}_2$, then $c = \{c^{cl}_1\}$.*
2. *If $c^{cl}_1 = \{-\}$ or if $c^{cl}_2 = \emptyset$, then $c = \{\{-\}\}$.*
3. *If $(d,\emptyset) \notin c^{cl}_2$ and $c^{cl}_2 \neq \emptyset$ and $c^{cl}_1 \neq \{-\}$, then c contains the following clauses: (a) for every dimension specifier $(d,V') \in c^{cl}_2$ such that $(d,V) \in c^{cl}_1$, the clause $\{(d,V\text{-}V')\} \cup \{(d_i,V_i) \mid (d_i,V_i) \in c^{cl}_1$ with $d_i \neq d\}$ belongs to c, and (b) for every dimension specifier $(d,V') \in c^{cl}_2$ such that $(d,V) \notin c^{cl}_1$, the clause $\{(d,\mathbf{V}_d\text{-}V')\} \cup \{(d_i,V_i) \mid (d_i,V_i) \in c^{cl}_1\}$ belongs to c.*

Note that if $c^{cl}_2 = \{-\}$ then, according to Definition 3.10, $(d,\emptyset) \in c^{cl}_2$ for every possible set of dimensions. Clause difference can be comprehended through the following example.

Example 3.13

A number of cases of clause difference are listed below. Reminder: the result of clause difference is a context specifier.

$\mathbf{V}_{lang}$ = {en, gr, sp}
$\mathbf{V}_{detail}$ = {low, medium, high}
$\mathbf{V}_{format}$ = {ps, pdf, html, doc}
$\mathbf{V}_{currency}$ = {Euro, USD}

- 1.
c^{cl}_1 = {(lang,{en,gr}),(detail,{low,high})}
c^{cl}_2 = {(lang,{en,gr}),(detail,{low})}
$c^{cl}_1 -^{cl} c^{cl}_2$ = {
{(lang,{}),(detail,{low,high})},
{(lang,{en,gr}),(detail,{high})} }
- 2.
c^{cl}_1 = {(lang,{en,gr}),(detail,{low,high}),(format,{pdf})}
c^{cl}_2 = {(lang,{en,gr}),(detail,{low})}
$c^{cl}_1 -^{cl} c^{cl}_2$ = {
{(lang,{}),(detail,{low,high}),(format,{pdf})},
{(lang,{en,gr}),(detail,{high}),(format,{pdf})} }
- 3.
c^{cl}_1 = {(lang,{en,gr}),(detail,{low})}
c^{cl}_2 = {(lang,{en,gr}),(detail,{low, high})}
$c^{cl}_1 -^{cl} c^{cl}_2$ = {
{(lang,{}),(detail,{low})},
{(lang,{en,gr}),(detail,{})} }
- 4.
c^{cl}_1 = {(lang,{en}),(detail,{low,medium,high}),(format,{pdf,ps})}
c^{cl}_2 = {(lang,{en}),(detail,{low}),(format,{pdf,html,doc})}
$c^{cl}_1 -^{cl} c^{cl}_2$ = {
{(lang,{}),(detail,{low,medium,high}),(format,{pdf,ps})},
{(lang,{en}),(detail,{medium,high}),(format,{pdf,ps})},
{(lang,{en}),(detail,{low,medium,high}),(format,{ps})} }
- 5.
c^{cl}_1 = {(lang,{en,gr}),(detail,{low,high})}

```
c^cl_2 = {(lang,{en}),(format,{ps,pdf}),(currency,{USD})}
c^cl_1 −^cl c^cl_2 = {
      {(lang,{gr}),(detail,{low,high})},
      {(lang,{en,gr}),(detail,{low,high}),(format,{html,doc})},
      {(lang,{en,gr}),(detail,{low,high}),(currency,{Euro})} }
```

♦

For cases covered by Definition 3.25-(3), the result of clause difference has as many clauses as exist dimension specifiers (d,V′) in c^{cl}_2. In cases (1) and (2) of Example 3.13, the first clauses of clause difference do not represent any world, and every clause of the results contain dimension specifiers that correspond exactly to dimension specifiers in c^{cl}_1. In both cases (1) and (2), the value of dimension `detail` in the corresponding (second) clause of the result is the conventional set difference of the corresponding values in c^{cl}_1 and c^{cl}_2. Note that dimensions that exist in c^{cl}_1 but not in c^{cl}_2 are simply transferred to the result, like `format` in case (2). Case (5) demonstrates the case where c^{cl}_2 contains dimensions that do not exist in c^{cl}_1. For each such dimension d, a clause is added to the result containing the difference from the domain of d, $\mathbf{V}_d$.

From Definition 3.25 it is easy to see that the following hold for the universal clause $\mathcal{U}^{cl}$ and the empty clause $\mathcal{E}^{cl}$:

$$c^{cl} -^{cl} \mathcal{U}^{cl} = \mathcal{E} \qquad \textbf{Formula 3.14}$$

$$c^{cl} -^{cl} \mathcal{E}^{cl} = \{c^{cl}\} \qquad \textbf{Formula 3.15}$$

$$\mathcal{E}^{cl} -^{cl} c^{cl} = \mathcal{E} \qquad \textbf{Formula 3.16}$$

Like the rest of clause operations, clause difference is defined in such a way as to have a direct correspondence to the set of worlds defined by the clauses; more specifically, it represents the difference of the sets of worlds defined by the involved clauses, as it is shown in the following proposition.

Proposition 3.13

If c^{cl}_1, c^{cl}_2 are context specifier clauses and $W_D(c^{cl}_1)$, $W_D(c^{cl}_2)$ are the sets of worlds w.r.t. **D** *that they represent, then $W_D(c^{cl}_1 -^{cl} c^{cl}_2) = W_D(c^{cl}_1) - W_D(c^{cl}_2)$.*

Proof: First, note that the $W_D(...)$ in the left side of the equality operates on a context specifier, while the two $W_D(...)$ in the right side operate on context specifier clauses. From Formula 3.14 we see that the claim holds if $c^{cl}_2 = \varnothing$ (universal clause $\mathcal{U}^{cl}$). In addition, from Formula 3.15 and Formula 3.16 the claim holds if $c^{cl}_2 = \mathcal{E}^{cl}$, or if $c^{cl}_1 = \{-\}$ (empty clause $\mathcal{E}^{cl}$). In what follows, we assume that c^{cl}_1 is not $\{-\}$, that c^{cl}_2 is nonempty, and that dimension specifiers in c^{cl}_2 are *not* of the form $(d,\varnothing)$. Those conditions are covered by case (3) of Definition 3.25. Clause difference as defined in case (3) of Definition 3.25, represents the worlds w.r.t. **D** that belong to c^{cl}_1 but not to c^{cl}_2, as we will show in what follows. Consider a clause that represents some of the worlds of c^{cl}_1. Such a clause would contain dimension specifiers for every dimension encountered in c^{cl}_1, because omission of such a dimension specifier (which is actually a constraint) would cause the inclusion of extra worlds, not represented by c^{cl}_1 (except for the trivial case where $(d,\mathbf{V}_d) \in c^{cl}_1$). Consequently, there are two ways a clause could represent a subset of the worlds of c^{cl}_1: (a) contain dimension specifiers (d,V′) where $(d,V_1) \in c^{cl}_1$ and $V' \subseteq V_1$, and (b) include additional dimension specifiers (d,V′) where $(d,V_1) \notin c^{cl}_1$. For the left out worlds to belong to c^{cl}_2, they must contain pairs (d,v) such that $v \in V_2$, where $(d,V_2) \in$

c^{cl}_2. Therefore, for any $(d,V_2) \in c^{cl}_2$ there are two cases that correspond to cases (a) and (b) above: (a) $(d,V_1) \in c^{cl}_1$, in which case a clause like c^{cl}_1, but with $(d, V_1 - V_2)$ instead of (d,V_1), represents all worlds in c^{cl}_1 and not in c^{cl}_2 as determined by dimension d, and (b) $(d,V_1) \notin c^{cl}_1$, in which case a clause like c^{cl}_1 but with the additional dimension specifier $(d,\mathbf{V}_d - V_2)$ would represent all worlds in c^{cl}_1 but not in c^{cl}_2 as determined by dimension d, since the dimension specifier $(d,\mathbf{V}_d)$ is implied in c^{cl}_1. All worlds represented by those clauses belong to c^{cl}_1 but do not belong to c^{cl}_2, and there is no world with these properties outside one of those clauses. The union of the worlds represented by such clauses (one clause for every dimension specifier in c^{cl}_2) will give all worlds in c^{cl}_1 that do not exist in c^{cl}_2. According to case (3) of Definition 3.25, those clauses form the result of $c^{cl}_1 -^{cl} c^{cl}_2$, and the union of the worlds represented by those clauses is $W_D(c^{cl}_1 -^{cl} c^{cl}_2)$, according to Formula 3.6.

Note that for cases (1) to (4) of Example 3.13, clause difference does not take into account any dimension domains, just like it happens with clause intersection and clause union. In case (5), however, we see that if c^{cl}_2 contains a dimension specifier for a dimension not contained in c^{cl}_1, then $\mathbf{V}_d$, the domain of d, must be taken into account. In order to keep $\mathbf{V}_d$ out of the picture, we could have introduced a special syntax to denote complement against $\mathbf{V}_d$, like for instance `(currency, not in {USD})`, and `(format, not in {ps,pdf})`. That approach would just transfer the problem to the definition of other operations, like clause intersection. In Section 3.6 we present a convenient notation for context that includes a similar syntax for dimension specifier complement. This however, is meant only as syntactic shorthand, because dimension domains are assumed to be finite and discrete, therefore substitution is always possible and complements can always be eliminated before performing context operations. Therefore, clause difference may occasionally need to take into account dimension domains (which are considered invariable, as stated in Section 3.2.1).

Like clause intersection and clause union, *clause difference is performed without taking into account any specific set of dimensions*. In other words, if $c^{cl}_1 -^{cl} c^{cl}_2 = c$ w.r.t. $\mathbf{D}$ and $c^{cl}_1 -^{cl} c^{cl}_2 = c'$ w.r.t. $\mathbf{D}'$, then c and c′ are identical. In addition, as shown in Proposition 3.13, c represents the worlds w.r.t. $\mathbf{D}$ covered by c^{cl}_1 but not by c^{cl}_2, and c′ represents the worlds w.r.t. $\mathbf{D}'$ covered by c^{cl}_1 but not by c^{cl}_2. Therefore, if $\mathbf{D}_{min}$ contains all the dimensions encountered in c^{cl}_1 and c^{cl}_2 and none other, the results of clause difference can be correctly interpreted w.r.t. any $\mathbf{D}$, with $\mathbf{D}_{min} \subseteq \mathbf{D}$.

Context Difference

Context difference $-^c$ is a binary operation that takes two context specifiers c_1, c_2 and returns a context specifier c_3.

$-^c$: (C x C) → C, where C is the set of all context specifiers

Context difference is defined below in terms of clause difference, context intersection, and context union.

Definition 3.26

*Let $c_1 = \{c^{cl}_{11}, c^{cl}_{12}, ..., c^{cl}_{1n}\}$, $c_2 = \{c^{cl}_{21}, c^{cl}_{22}, ..., c^{cl}_{2m}\}$ where n, m ≥ 1, be context specifiers. The **context difference** of c_1 and c_2, denoted $c_1 -^c c_2$, is a context specifier c_3 defined as the context union of all context specifiers $[(c^{cl}_{1k} -^{cl} c^{cl}_{21}) \cap^c (c^{cl}_{1k} -^{cl} c^{cl}_{22}) \cap^c ... \cap^c (c^{cl}_{1k} -^{cl} c^{cl}_{2m})]$ where $c^{cl}_{1k} \in c_1$, $n \geq k \geq 1$.*

*The context specifier given by $\{\varnothing\} -^c c$ is called **context complement** of c.*

The set of worlds w.r.t. **D** represented by the context difference of two contexts is equal to the difference of the sets of worlds represented by the contexts:

Proposition 3.14

For two context specifiers c_1, c_2 the following holds: $W_D(c_1 -^c c_2) = W_D(c_1) - W_D(c_2)$.

Proof: Lets assume that $c_1 = \{c^{cl}_{11}, c^{cl}_{12}, ..., c^{cl}_{1n}\}$ and $c_2 = \{c^{cl}_{21}, c^{cl}_{22}, ..., c^{cl}_{2m}\}$ where n, m ≥ 1. From Formula 3.6, we have that $W_D(c_1) - W_D(c_2) = [W_D(c^{cl}_{11}) \cup W_D(c^{cl}_{12}) \cup ... \cup W_D(c^{cl}_{1n})] - [W_D(c^{cl}_{21}) \cup W_D(c^{cl}_{22}) \cup ... \cup W_D(c^{cl}_{2n})] = [(W_D(c^{cl}_{11}) - W_D(c^{cl}_{21})) \cap ... \cap (W_D(c^{cl}_{11}) - W_D(c^{cl}_{2m}))] \cup ... \cup [(W_D(c^{cl}_{1n}) - W_D(c^{cl}_{21})) \cap ... \cap (W_D(c^{cl}_{1n}) - W_D(c^{cl}_{2m}))]$. From Proposition 3.13, the last expression becomes: $[W_D(c^{cl}_{11} -^{cl} c^{cl}_{21}) \cap ... \cap W_D(c^{cl}_{11} -^{cl} c^{cl}_{2m})] \cup ... \cup [W_D(c^{cl}_{1n} -^{cl} c^{cl}_{21}) \cap ... \cap W_D(c^{cl}_{1n} -^{cl} c^{cl}_{2m})]$. Because of Proposition 3.10, the last expression becomes: $W_D[(c^{cl}_{11} -^{cl} c^{cl}_{21}) \cap^c ... \cap^c (c^{cl}_{11} -^{cl} c^{cl}_{2m})] \cup ... \cup W_D[(c^{cl}_{1n} -^{cl} c^{cl}_{21}) \cap^c ... \cap^c (c^{cl}_{1n} -^{cl} c^{cl}_{2m})]$, and from Proposition 3.12: $W_D([(c^{cl}_{11} -^{cl} c^{cl}_{21}) \cap^c ... \cap^c (c^{cl}_{11} -^{cl} c^{cl}_{2m})] \cup^c ... \cup^c [(c^{cl}_{1n} -^{cl} c^{cl}_{21}) \cap^c ... \cap^c (c^{cl}_{1n} -^{cl} c^{cl}_{2m})])$. From Definition 3.26 the last expression becomes $W_D(c_1 -^c c_2)$.

From Proposition 3.14, the following hold for universal contexts $\mathcal{U}$ and empty contexts $\mathcal{E}$:

$c -^c \mathcal{U} = \mathcal{E}$ **Formula 3.17**

$c -^c \mathcal{E} = c$ **Formula 3.18**

$\mathcal{E} -^c c = \mathcal{E}$ **Formula 3.19**

Note that, in Formula 3.18 conventional equality is used instead of context equality that is implied by Proposition 3.14. Formula 3.18 is derived from Definition 3.26 and from Formula 3.15.

From Definition 3.26 we see that a context difference $c_1 -^c c_2$ is defined in terms of context intersection, context union, and clause difference, which are performed without taking into account any specific set of dimensions. Therefore, if $\mathbf{D}_{min}$ contains all the dimensions encountered in c_1 and c_2 and none other, the results of context difference can be correctly interpreted w.r.t. any **D**, with $\mathbf{D}_{min} \subseteq \mathbf{D}$.

3.3.2.8 Context Subset

Context subset $\subseteq^c$, and **proper context subset** $\subset^c$ are binary operations that take two context specifiers c_1, c_2 and return a Boolean value.

```
⊆c : (C x C) → Boolean, where C is the set of all contexts
⊂c : (C x C) → Boolean, where C is the set of all contexts
```

Context subset / superset and proper context subset / superset are defined as follows:

Definition 3.27

*Let **D** be a nonempty set of dimension names, and for each $d \in \mathbf{D}$ let $\mathbf{V}_d$ be the domain of d, with $\mathbf{V}_d \neq \varnothing$. Let c_1, c_2 be context specifiers. Then c_1 is **context subset** of c_2, denoted $c_1 \subseteq^c c_2$, or equivalently c_2 is **context superset** of c_1, denoted $c_2 \supseteq^c c_1$, iff $W_D(c_1) \subseteq W_D(c_2)$, where $W_D(c)$ is the set of worlds w.r.t. **D** represented by c.*

*If $W_D(c_1) \subset W_D(c_2)$, then c_1 is **proper context subset** of c_2, denoted $c_1 \subset^c c_2$, or equivalently c_2 is **proper context superset** of c_1, denoted $c_2 \supset^c c_1$.*

From Definition 3.27, it is possible to determine whether a context specifier is (proper) context subset w.r.t. **D** of another context specifier, by comparing the corresponding sets of worlds w.r.t. **D**. In addition, context subset can be expressed in terms of context intersection and context equality, exactly as it happens with conventional sets.

Proposition 3.15

For two context specifiers c_1, c_2 the following hold: (a) $c_1 \subseteq^c c_2 \Leftrightarrow c_1 \cap^c c_2 =^c c_1$, and (b) $c_1 \subset^c c_2 \Leftrightarrow (c_1 \cap^c c_2 =^c c_1$ AND $c_1 \neq^c c_2)$.

Proof: Consider that the left side of expression (a) is true. Then, $W_D(c_1) \subseteq W_D(c_2) \Leftrightarrow W_D(c_1) \cap W_D(c_2) = W_D(c_1)$ and from Proposition 3.10 $W_D(c_1 \cap^c c_2) = W_D(c_1)$. From Definition 3.19 we have that $c_1 \cap^c c_2 =^c c_1$. In the same way if the right side of expression (a) holds, then the left side holds as well. Expression (b) is obvious from the definitions of the operations involved.

Although context (proper) subset / superset are defined by referring to some set of dimensions **D**, they are not actually defined with respect to any particular **D** [21]. From Proposition 3.15, and because context intersection and context equality do not depend on **D**, it is easy to see that, if $\mathbf{D}_{min}$ contains all the dimensions encountered in c_1 and c_2 and none other, then $c_1 \subseteq^c_D c_2 \Leftrightarrow c_1 \subseteq^c_{D'} c_2$ and $c_1 \subset^c_D c_2 \Leftrightarrow c_1 \subset^c_{D'} c_2$, $\forall$ **D** and **D′** with $\mathbf{D}_{min} \subseteq \mathbf{D}$ and $\mathbf{D}_{min} \subseteq \mathbf{D'}$.

Concerning the empty context and the universal context, it is obvious that $\mathcal{E} \subseteq^c c$ and $c \subseteq^c \mathcal{U}$, for any context specifier c.

3.4 PROPERTIES OF CONTEXT OPERATIONS

In Section 3.3 we introduced a number of operations on context specifier clauses and on context specifiers, and we established the relationship between contexts and sets of worlds. We have also stated some of the properties of context operations, specifically those that involve the empty contexts $\mathcal{E}$ and the universal contexts $\mathcal{U}$. In this section we investigate the properties of context operations in more detail.

Context specifiers have an important property not exhibited by clauses, i.e. they are *closed* under the operations of context intersection, context union, and context difference: from the definitions of those operations it is easy to see that their result is always a context specifier. This is also the case for the expanded form of a context specifier, which is also a context specifier. In addition, given a finite **D** and $\mathbf{V}_d$ for every d in **D**, if the context specifiers involved are finite, then the result of context operations (including context extension) will also be finite. In what follows we assume finite context specifiers.

[21] However, checking two context specifiers for context (proper) subset / superset may need to take into account the domains of the dimensions encountered in those contexts.

3.4.1 Dependence on Dimensions

Context operations have been separated in two groups. The first group consists of operations that are performed without taking into account any specific set of dimensions, namely context extension, context intersection, context union, context difference, context equality / inequality, and (proper) context subset / superset. The second group consists of those operations that are performed with respect to a specific set of dimensions, and are marked with the symbol $_D$ under the operation sign, namely context expansion, and context expanded extension.

In what follows, we set aside the operations used for transforming contexts into sets of worlds, namely context expansion, context extension, and context expanded extension. We focus on context equality / inequality, (proper) context subset / superset, context intersection, context union, and context difference, which are used to form **context expressions** and **context conditions** (introduced in Section 3.4.4). These operations give the same results with respect to different sets of dimensions. Specifically, we have seen that they can be performed with respect to $\mathbf{D}_{min}$, which contains only those dimensions encountered in the contexts involved in the operation. Then the result will be correctly interpreted w.r.t. any $\mathbf{D}$ that is superset of $\mathbf{D}_{min}$.

This leads to some useful conclusions for context equality / inequality, (proper) context subset / superset, context intersection, context union, and context difference:

(a) Adding dimensions to any given $\mathbf{D}$ does not affect the results of operations performed w.r.t. $\mathbf{D}$.

(b) It is possible to perform context operations without knowing what the actual set of dimension $\mathbf{D}$ is at that time.

(c) Context equality / inequality and (proper) context subset / superset can consider $\mathbf{D}_{min}$ instead of the actual $\mathbf{D}$ to enhance efficiency.

Context intersection and context union can consider the domain of any dimension d to be $\mathbf{V}_{min\text{-}d}$, the set of all values of d encountered in the contexts involved in the operation. It is easy to realize that the results of those operations will also hold for any $\mathbf{V}_d$, where $\mathbf{V}_d$ is superset of $\mathbf{V}_{min\text{-}d}$. Therefore, it is not necessary to know the dimension domains for performing context intersection and context union. On the contrary, context difference is an operation that needs to take into account the actual domains of dimensions, since its result may[22] depend on dimension domains. For the same reason[23], context equality / inequality and (proper) context subset / superset must take into account the actual dimension domains as well.

3.4.2 Context Equality and Context Subset

In Section 3.3 we have used two types of equality with context specifiers, namely conventional set-based equality and context equality. Conventional set-based equality is not

[22] As explained in Section 3.3.2.7, if a clause in c_2 contains the dimension specifier (d,V_2) and some clause in c_1 does not contain a corresponding dimension specifier (d,V_1), then $c_1 -^c c_2$ will have to take into account the actual domain $\mathbf{V}_d$ of d.

[23] The result of context equality / inequality depends on dimension domains: {{(lang,{gr,en})}} is context equal to {{(detail,{low,high})}} if $\mathbf{V}_{lang}$ = {gr,en} and $\mathbf{V}_{detail}$ = {low,high}, but not context equal in any other case. The same is the case for (proper) context subset / superset, which can be expressed in terms of context intersection and context equality / inequality, as showed in Section 3.3.2.8.

very useful, because it compares the form of two contexts and not the worlds they represent. Context equality on the other hand is defined based on the sets of worlds represented by contexts. Conventional equality is more restrictive than context equality. In other words, the following holds:

$$c_1 = c_2 \quad \Rightarrow \quad c_1 =^c c_2$$

Note that the opposite entailment (to the left) does not hold.

From its definition it is evident that context equality is:

(a) Reflexive, $c_1 =^c c_1$.

(b) Symmetric, $c_1 =^c c_2 \Rightarrow c_2 =^c c_1$.

(c) Transitive, $(c_1 =^c c_2, c_2 =^c c_3) \Rightarrow c_1 =^c c_3$.

Because of properties (a), (b), and (c), context equality is an equivalence relation [SR86], and defines equivalence classes on the set of all context specifiers. Specifically, for any context specifier c, the equivalence class of c relative to context equality is the set of context specifiers c′ such that $c =^c c'$. Such an equivalence class contains all contexts that represent the same set of worlds w.r.t. some (any) **D**.

Likewise, (proper) context subset / superset have properties similar to the properties of (proper) subset / superset of conventional sets.

3.4.3 Context Intersection, Context Union, and Context Difference

As we have showed, those operations correspond to the conventional set operations of intersection, union, and difference if we view context specifiers as the set of worlds they represent. It is easy to prove that properties of conventional set operations for intersection, union, and difference hold for the corresponding context operations as well. As an example, we give the proof of the associative law for context intersection. Note that context equality is used instead of conventional set equality.

Proposition 3.16

Context intersection is associative: $c_1 \cap^c (c_2 \cap^c c_3) =^c (c_1 \cap^c c_2) \cap^c c_3$

Proof: Both the left and right sides give context specifiers. For those to be context equal, they must represent the same set of worlds w.r.t. some **D**. Note however that according to Proposition 3.10, $W_D[c_1 \cap^c (c_2 \cap^c c_3)] = W_D(c_1) \cap [W_D(c_2) \cap W_D(c_3)]$. However because of the associative property of conventional set intersection, the last expression is equal to $W_D[(c_1 \cap^c c_2) \cap^c c_3]$.

An indicative list of basic properties for context intersection and context union is given below.

- Idempotent:

$$c \cap^c c =^c c$$
$$c \cup^c c =^c c$$

- Associative:

$$c_1 \cap^c (c_2 \cap^c c_3) =^c (c_1 \cap^c c_2) \cap^c c_3$$
$$c_1 \cup^c (c_2 \cup^c c_3) =^c (c_1 \cup^c c_2) \cup^c c_3$$

- Commutative:

$c_1 \cap^c c_2 =^c c_2 \cap^c c_1$
$c_1 \cup^c c_2 =^c c_2 \cup^c c_1$

- Distributive:

$c_1 \cap^c (c_2 \cup^c c_3) =^c (c_1 \cap^c c_2) \cup^c (c_1 \cap^c c_3)$
$c_1 \cup^c (c_2 \cap^c c_3) =^c (c_1 \cup^c c_2) \cap^c (c_1 \cup^c c_3)$

In addition, it is easy to see that, like conventional intersection and union, context intersection and context union are *monotone*: if $c_1 \subseteq^c c_1'$ and $c_2 \subseteq^c c_2'$, then $(c_1 \cap^c c_2) \subseteq^c (c_1' \cap^c c_2')$, and $(c_1 \cup^c c_2) \subseteq^c (c_1' \cup^c c_2')$.

Likewise, context difference has properties similar to the properties of conventional set difference.

3.4.4 Context Expressions and Context Conditions

Context expression is an expression comprised of context specifiers, joined together through context intersection, context union, or context difference, with brackets defining precedence for context operations.

Context condition is a condition comprised of context expressions compared through context equality / inequality, or (proper) context subset / superset, combined with "`and`" or "`or`", and qualified with "`not`", with brackets defining precedence.

Replacing a context specifier with a context equal context specifier yields an equivalent context expression (an expression representing the same set of worlds as the original), or context condition. Context expanded form and simplified forms (as discussed in Section 3.3.2.4) of a context specifier c are context equal to c, and can substitute c in context expressions and context conditions.

3.5 COMPLEXITY OF CONTEXT OPERATIONS

Time complexities of context operations are discussed in this section, based on the number of comparisons performed. We assume that the size of the problem is represented by the number of clauses *k* of context specifiers. This number can be greater than the number of worlds, since overlapping clauses are allowed. Furthermore, operations like context intersection, context union, and context difference increase the number of clauses contained in context specifiers[24].

For the following we assume a set of dimensions **D** with cardinality *n*, and we assume that the mean cardinality of the dimension domains is *m*. Then, the number of worlds defined by **D** is m^n. We also assume that context specifiers are ordered in the following manner: (a) dimension values in dimension specifiers are ordered, according to their numeric value or alphabetically, and (b) dimension specifiers inside clauses are ordered according to the lexicographical ordering of dimension names.

[24] In Section 3.3.2.4, we showed how the number of clauses can be reduced by simplifying context specifiers.

Context intersection, $c_1 \cap^c c_2$: We assume that the mean number of dimension specifiers (d,V) in the clauses of c_1 and c_2 is p ($p \leq n$), and that the mean number of dimension values in V is v ($v \leq m$). Clause intersection between two clauses will first compare dimension specifiers and if the dimension d is the same, the dimension value sets will be compared. Therefore, $p+p*v = p*(v+1)$ number of comparisons will be performed at the worst case. If c_1 contains k clauses and c_2 contains l clauses, then the estimated total number of comparisons at the worst case is $k * l * p*(v+1)$. However, p and v are upper bounded, since in the frame of any given problem the set of dimensions and the corresponding domains are finite. If we consider k to be the mean number of clauses in c_1 and c_2, then the order of time complexity at the worst case is $O(k^2)$.

Context union, $c_1 \cup^c c_2$: Context union does not perform any comparisons, and the order of time complexity is O(1).

Context expanded extension, $\otimes_D$ c: If c contains k number of clauses then the expansion will perform $n*k$ number of comparisons at the worst case. The extension stage will not perform any comparisons, therefore the order of time complexity at the worst case is $O(k)$.

Elimination of duplicate worlds will increase the number of comparisons. In particular, the elimination of duplicate worlds requires at most $n*w$ comparisons for each pair of clauses in c, where w is the mean number of worlds represented by the clauses of c, $w \leq m^n$. The total number of comparisons for context expanded extension with elimination of duplicates at the worst case will be $n*k+n*w*(1+2+...+k-1)$, therefore[25] the order of time complexity is $O(k+(k-1)^2)$, or more simply $O(k^2)$.

The factor $(k-1)^2$ corresponding to the extra cost for eliminating duplicate worlds can be avoided, if c contains clauses that are mutually exclusive. Context intersection preserves this quality, in other words if c_1 and c_2 contain only mutually exclusive clauses, then $c = c_1 \cap^c c_2$ contains only mutually exclusive clauses as well. Maintaining contexts in such a form requires to properly reform c, if c is the result of a context expression that involves context unions and / or context differences.

Context equality, $c_1 =^c c_2$: We assume that the least number of worlds represented by c_1 and c_2 is w, $w \leq m^n$. Then, the comparisons at the worst case are $n*w$, plus the comparisons needed for context expanded extension for both c_1 and c_2. Therefore, the order of the time complexity at the worst case is the same as in context expanded extension: $O(k^2)$ if c_1, c_2 contain overlapping clauses, where k is the mean number of clauses in c_1 and c_2. If clauses in each of c_1, c_2 are mutually exclusive, the order of time complexity for context equality at the worst case is $O(k)$.

Context subset, $c_1 \subseteq^c c_2$: Same as in context equality.

The above discussion leads us to consider context specifiers containing clauses that are mutually exclusive with one another. In this case, a context will not contain more clauses than there are worlds. If $w = m^n$ is the total number of possible worlds, and assuming context specifiers that consist of mutually exclusive clauses, the order of the time complexity in the worst case is as follows.

Context intersection remains $O(w^2)$.

Context union: $O(w^2)$.

Context expanded extension, *context equality* and *context subset*: $O(w)$.

[25] Using that $1+2+...+k = O(k^2)$.

Notice that the cost of maintaining context specifiers in a state where clauses are mutually exclusive falls on context union and context difference, which must compare and rearrange clauses.

As stated in Section 3.4.1, the efficiency of context equality / inequality and (proper) context subset / superset can be enhanced by considering $\mathbf{D}_{min}$ as the set of dimensions, where $\mathbf{D}_{min}$ contains all the dimensions encountered in the contexts involved in the operation and none other.

3.6 A NOTATION FOR CONTEXT SPECIFIERS

In previous sections we formally defined context specifiers and context operations. In this section we introduce a simpler and more convenient syntax for context specifiers and context operations.

Example 3.14
The following are examples of context specifiers:

```
1. [time=07:45]
2. [lang=greek, detail in {low,medium}]
3. [lang not in {greek,spanish}, detail!=high]
4. [season=summer | season in {fall,spring}, daytime=noon]
5. [time in {08:00..13:30,17:00..20:30}]
6. [format=pdf | | - | lang!=spanish]
7. []
8. [-]
```

♦

In Example 3.14, context specifier (1) consists of one clause containing just one dimension specifier, and is formally written as {{(time,{07:45})}}. Context specifier (2) consists of a clause that contains two dimension specifiers, one for dimension `lang` and one for dimension `detail`. Context specifier (3) uses "`not in`" and "`!=`" (not equal), which do not have any counterpart in the formal definition of dimension specifiers. However, this is presented just as a syntactic shorthand, because we assume that dimension domains are known at the time contexts are used. In addition, as stated in Section 3.2.1, we assume finite dimension domains, which can be described by enumerating their elements. It is, therefore, always possible to find the set complement implied by the operators in (3), and express contexts in a way that has a formal counterpart. Context specifier (4) contains two clauses, and represents the worlds where it is either summer or fall / spring noon. Context specifier (5) shows a shorthand for representing intervals over a bounded, discrete, and totally ordered dimension domain. Context specifier (6) consists of four clauses; the second clause does not contain anything and corresponds to the universal clause ∅, while the third clause is the symbol "–" and corresponds to the empty clause {–}. Finally, the formal counterpart of context specifier (7) is {∅}, a universal context for any set of dimensions, and the formal counterpart of context specifier (8) is {{–}}, an empty context for any set of dimensions.

The grammar for context specifiers is given in Table 3.1 below in Extended Backus-Naur Form [EBNF], or EBNF for short. Symbols that can be defined by a regular expression start with a capital letter (example: `DimValue`), while symbols defined in EBNF start with a lowercase letter (example: `cxtSpec`).

Table 3.1: Syntax of context specifiers.

```
cxtSpec ::= "[" cxtSpecClause ("|" cxtSpecClause)* "]"
cxtSpecClause ::= "" | "–" | dimList
dimList ::= dimSpec ("," dimSpec)*
dimSpec ::= dimName (atomicOp DimValue |
                     setOp ("{" setDimValue? "}" | "ALL"))
atomicOp ::= "=" | "!="
setOp ::= "in" | "not in"
setDimValue ::= dimSpan ("," dimSpan)*
dimSpan ::= DimValue | DimValue ".." DimValue
```

As one would expect, it is legal for context specifiers to contain empty sets, as in `[lang=english, detail in {}]` and `[lang=english, detail not in {}]`. In the latter case, it is implied that `detail` ranges over its complete domain. In addition, the keyword `ALL` is shorthand for the complete domain of a dimension, like for instance in `[time in ALL]`[26]. Once more, context specifiers containing "`ALL`", "`not in`", or "`!=`" are eventually substituted by context equal context specifiers that do not contain those operators. Consequently, in such cases the dimension domains must be taken into account before performing any operation[27].

The notation for context operations is given in Table 3.2 that follows.

Table 3.2: A notation for context operations.

Operation	Symbol	Notation
Context equality	$=^c$	=
Context inequality	$\neq^c$	!=
Context subset, superset	$\subseteq^c$, $\supseteq^c$	<= , >=
Proper context subset, superset	$\subset^c$, $\supset^c$	< , >
Context intersection	$\cap^c$	*
Context union	$\cup^c$	+
Context difference	$-^c$	–

Using this notation, it is possible to form context expressions and context conditions like the following:

- `[lang=greek, detail in {low,medium}] <= [lang=greek]`
- `[format=pdf | detail!=high] * [lang=english, format=pdf] = [lang=english, format=pdf]`

[26] Obviously, a dimension specifier of the form "`d in ALL`" can be completely omitted from a clause. Despite that, the keyword `ALL` is still useful, as we shall see in the following chapters where we discuss **context patterns**.

[27] This includes context intersection and context union, which otherwise do not depend on dimension domains.

3.7 SUMMARY

Contrary to traditional databases, information on the Web is often associated with an underlying *context*. In this chapter we developed a formalism for representing context in a flexible and intuitive way, using variables called **dimensions**. **Context specifiers** define contexts by constraining the possible values dimensions can take, and by combining such dimension constraints in conjunctions and disjunctions.

We gave semantics to context by showing that it can be viewed as a set of possible **worlds**, where a world is an environment under which information obtains an unambiguous meaning. We defined a number of context operations for combining and comparing contexts, and proved that those operations observe a correspondence to conventional sets of possible worlds.

Moreover, we investigated the properties of context operations, and discussed their time complexities. An important advantage of our approach is that most context operations can be performed even if our knowledge of the dimensions involved is incomplete; in this case the correspondence of contexts to sets of worlds can be deferred until knowledge of dimensions is complete. Finally, we proposed a convenient notation for expressing contexts and context operations.

4 A DATA MODEL FOR MSSD

In the previous chapter we introduced context specifiers as a way to represent context using multiple dimensions, explained their semantics through correspondence to worlds, and defined a number of context operations used to formulate context expressions and context conditions. Context specifiers are the key difference between conventional data and context-dependent multifaceted data, which are also called **multidimensional data** in our approach. Therefore, context specifiers play a central role in data models and query languages for multidimensional data.

In this chapter we propose a data model for multidimensional semistructured data (MSSD), called **Multidimensional Object Exchange Model** (also called **Multidimensional OEM**, or **MOEM** for short). MOEM is a particular case of a graph-based data model called **Multidimensional Data Graph**, which incorporates context specifiers and uses special nodes and edges to represent **multidimensional entities**. Following the semistructured culture according to which schema exists as part of data, we embed context information in the graph model itself. MOEM and Multidimensional Data Graph extend *Object Exchange Model* [PGW95, AQM+97, Suc98], or *OEM*, a predominant graph-based model for semistructured data, originally designed at Stanford University as part of the TSIMMIS project [PGW95, GPQ+97, CGH+94, GHI+95, PGU96]. As we will show, MOEM consolidates a number of different OEMs holding under different worlds.

The need for an OEM extension has also been recognized in [CAW99], where the problem of representing histories of changes in OEM databases is addressed. In Chapter 6 we discuss [CAW99] further, and we investigate the aforementioned problem in detail. A model for semistructured data that deviates from OEM has been proposed in [BDT98], where edge labels are themselves pieces of semistructured data. The model we propose views labels as containing metadata rather than data. In [DBJ99], an extensible semistructured data model has been proposed that uses sets of properties of the form `property_name: property_value` as edge labels. By attaching properties at will, the graph becomes rich in metadata. Different properties may have different semantics, which results in a model of increased generality, but on the other hand makes the formulation of queries more complicated. The goal of our approach is to represent information that presents different facets. This leads to common semantics for the metadata, which is used solely to embody context information. In addition, our model retains OEM labeled edges and attaches context metadata to a new type of edges; graph models such as OEM become special cases of the model we propose.

In what follows, we define the notion of a multidimensional entity, and explain how context specifiers are used within multidimensional entities. We then proceed to Multidimensional Data Graph, and we investigate its properties: propagation of context, reduction, context-determinism, and canonical form. Based on those properties, we define Multidimensional OEM as a special case of Multidimensional Data Graph. Finally, we introduce mssd-expressions, a way to textually represent Multidimensional Data Graphs and MOEMs.

4.1 MULTIDIMENSIONAL ENTITIES

In multidimensional data, context is used to qualify **facets** of the same information entity, stating the conditions under which each facet holds and, at the same time, providing the frame for interpreting information in an unambiguous way. An information entity that is comprised of a number of facets, each holding under a number of worlds, is called **multidimensional entity**.

Definition 4.1

*A **multidimensional entity** is an information entity that encompasses zero or more **facets** together with corresponding contexts that define the worlds under which each facet may hold. The facets of a multidimensional entity may themselves be multidimensional entities, or conventional [28] information entities, or any combination of the two.*

Multidimensional entities are the cornerstone of multidimensional data, since they (a) encompass context specifiers, which are confined to appear only within multidimensional entities, and (b) keep together information entities that constitute facets of the same abstract entity. A multidimensional entity may have any number of facets (no facets at all in the trivial case), and facets that are conventional information entities may have different values[29]. Note that *conventional information entities can be considered as a special case of multidimensional entities*, where there exists only one facet holding under every possible world (the facet context is a universal context $\mathcal{U}$).

There are no restrictions on what can be the respective context of a facet; facets of a multidimensional entity may be qualified by contexts that overlap (define common worlds), or even by empty contexts in the trivial case. Thus, a multidimensional entity is not obliged to cover every possible world through its facets: given a world w, zero, one, or more facets of a multidimensional entity may hold under w.

Given a set of worlds S_W, there are two principal ways we can select facets of a multidimensional entity[30]. The first is to select a facet only if it holds under every world in S_W, thus to assume that a conjunction is implied for worlds in S_W. The second is to assume a disjunction, and to select a facet if it holds under at least one world that belongs to S_W. If a facet holds under every world represented by a context c, then we say that this facet **holds under** c.

Definition 4.2

*Let c be a context specifier, and let e be a multidimensional entity. Let f_i be a facet of e, and let c_i be a context specifying the worlds under which f_i holds. Then we say that f_i **holds under** c, iff $c \subseteq^c c_i$.*

Given a context c and a multidimensional entity e, if f_k, f_l,..., f_m are the facets of e that hold under c, then we say that e **evaluates to** f_k, f_l,..., f_m **under** c. Given a context, a multidimensional entity may evaluate to zero, one, or more facets. Note that in the trivial case where a multidimensional entity has no facets, it never evaluates to a facet.

---

[28] By "conventional" it is meant not context-dependent.

[29] As we shall see later in this section, in the case of multidimensional semistructured data, facets of a multidimensional entity may also have types (structure).

[30] Those ways are used in Section 4.4 for defining **reduction to OEM** and **partial reduction**, respectively.

As already stated, the facets of a multidimensional entity may have different values. Specifically in MSSD, *facets of a multidimensional entity may also have different types* (in other words, different structure). This is in line with the loose typing of (conventional) semistructured data, where instances of the same class of objects may exhibit varying structure.

4.2 MULTIDIMENSIONAL DATA GRAPHS

In this section we present *Multidimensional Data Graph*, a data model that extends OEM by incorporating context specifiers. Although OEM and similar graph-based Web data models are in principle capable of representing multidimensional entities, they fall short for a number of reasons.

(a) Hidden semantics: by not addressing directly the issue of multiple facets, it is the responsibility of an application to assign semantics in an ad-hoc manner.

(b) Cumbersome notation: representing multiple facets of an entity cannot be done in an elegant way.

(c) Duplication of information: an ad-hoc approach may lead to duplicating information that is common, which is undesirable.

Multidimensional Data Graphs avoid those shortcomings *by treating multidimensional entities as first class citizens*. Special nodes and edges are used to model multidimensional entities and to distinguish them from conventional ones. In addition, the notation for context we introduced in the previous chapter is used for qualifying facets of multidimensional entities.

As we have seen in Chapter 2, OEM is a *rooted directed labeled multigraph*, flexible enough to tolerate the irregularities of semistructured data. A number of OEM variants have been proposed that attach labels to either nodes [PGW95, GPQ+97] or edges [MAG+97, AQM+97, FFLS97, BFS00]. The question of whether labels should be attached to nodes or to edges is a recurring one in graph models for semistructured data and XML, and each approach gives its own answer. If labels are attached to nodes, then the label becomes a property of an object (since nodes represent objects), while if labels are attached to edges, then the label becomes a property of the relation between two objects. Attaching labels to edges allows nodes to "see" the same node through different names. In what follows, when we refer to OEM we assume that labels are attached to edges, which is also the approach taken by Multidimensional Data Graphs. As a formal definition of OEM, we adopt the one given in [Suc98] (see Chapter 2).

4.2.1 Why Start from OEM?

An essential component of any MSSD model must be context specifiers. Context specifiers are quite independent from a particular data model, thus any existing data model for semistructured data can be considered as an infrastructure for incorporating context specifiers. However, for investigating the properties of MSSD at an abstract level, a model that is expressive and simple must be selected. Expressive, as it must capture all the essential aspects of semistructured data. Simple, as it must use the least possible number of fundamental concepts, so that extending it with new concepts will not entail unnecessary complications.

XML [XML, Wal97, Cha99] has gained a wide acceptance, both as an exchange format and as a logical model for Web data; therefore XML is a strong candidate for incorporating context

specifiers. However, XML has a number of syntax particularities that complicates the incorporation of context specifiers. Specifically, an element in an XML document can relate to another element in two different ways: (a) it can include the element directly, or (b) it can reference it using identifiers through attributes of the special types "IDREF" and "IDREFS". The two ways are syntactically different, and this difference is often reflected in XML graph data models. However, both ways basically describe an element-subelement relationship. The reason for having two ways for describing essentially the same thing is that in an XML document an element cannot be included by more than one parent elements. Therefore, the syntactic mechanism of "IDREF" attributes is needed to transform the XML document tree into a more flexible graph, and to allow an element to be shared by many parent elements.

It would be desirable to select a model that is purged from such complications, to form the basis for investigating the properties of MSSD. The reasons for selecting OEM as the starting point is that it is both expressive and simple, and that it represents information in a uniform and consistent way. This uniformity makes easier the incorporation of new features like context specifiers, because one is able to concentrate on how the new features affect a small number of fundamental concepts, rather than worrying about secondary syntactic issues. In Chapter 7 we introduce *Multidimensional XML* (*MXML* for short) that consolidates context and multidimensional entities with XML.

4.2.2 Representing Multidimensional Entities

To model multidimensional entities, we introduce two new basic graph elements:

- **Multidimensional nodes**: a multidimensional node represents a multidimensional entity, and is used to group together nodes that constitute facets of that entity. Multidimensional nodes have a rectangular shape to distinguish them from conventional circular nodes.
- **Context edges**: context edges are directed labeled edges that connect a multidimensional node to its facets. The label of a context edge pointing to a facet *f* is a context specifier that defines the set of worlds under which *f* holds. Context edges are drawn as thick or double lines, to distinguish them from conventional edges drawn as single thin lines.

Facets that are not themselves multidimensional entities are represented as conventional, OEM-style circular nodes, which are called **context nodes**. All nodes in Multidimensional Data Graph, context or multidimensional, are considered objects and have a unique *object identifier* (*oid* for short). Like in OEM, oids by convention are preceded by the character *&*. In what follows, the terms **node** and **object** will be used interchangeably.

Figure 4.1: Graphical representation of multidimensional entities.

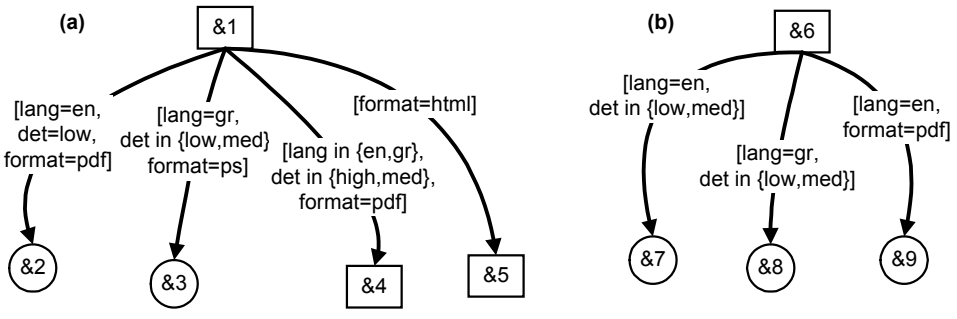
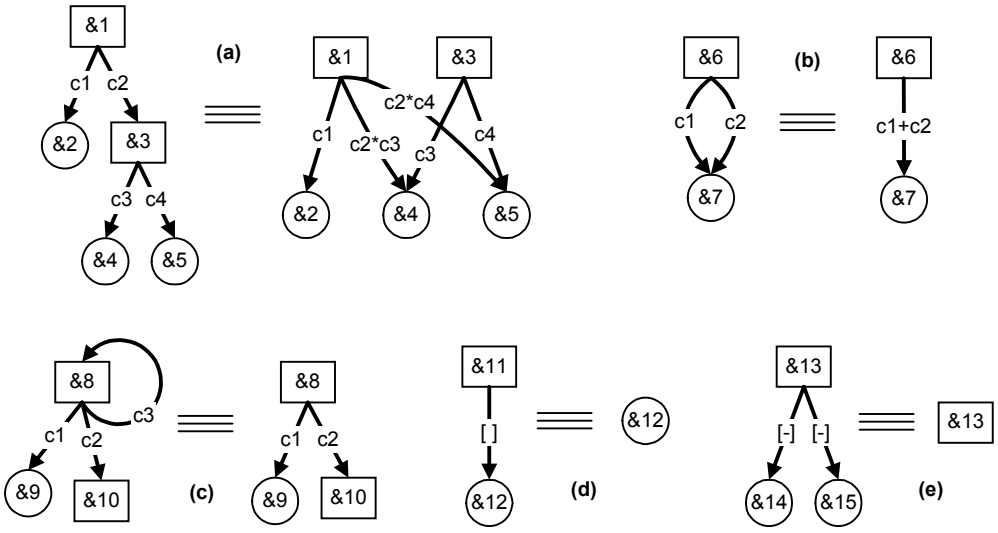


Figure 4.1 depicts[31] a couple of multidimensional entities denoted (a) and (b). The multidimensional node &1 in (a) represents a multidimensional entity whose facets comprise two context nodes (&2 and &3) and two multidimensional nodes (&4 and &5). These facets hold under disjoint sets of worlds, in other words, given any world w at most one facet holds under w. On the other hand, facets of the multidimensional entity in (b) comprise nodes &7 and &9, which hold under common worlds.

Given the context specifier c_1 = `[lang=en, det in {med, high}, format=pdf]`, the multidimensional entity in Figure 4.1 (a) evaluates to node &4 under c_1, and node &4 is also the only facet that holds under any world covered by c_1. Given the context specifier c_2 = `[lang=en, det in {med, high}, format=pdf | lang=gr]`, which is context superset of c_1, the multidimensional entity in (a) does not evaluate to any nodes under c_2, while facets &3, &4, &5 hold under worlds covered by c_2.

As we have seen in the previous chapter, context specifiers may have different forms but represent the same set of worlds. Consequently, two multidimensional entities may comprise different context specifiers, but be essentially the same as long as the respective context specifiers are context equal. In addition, multidimensional entities can be represented graphically in a number of alternative ways, while at the same time *preserving the worlds under which each facet holds*.

Figure 4.2: Alternative ways of representing multidimensional entities.



In Figure 4.2 we see some pairs of "equivalent"[32] multidimensional entities. Pair (a) shows that *nested* multidimensional entities can be expressed as *flat* multidimensional entities, where all facets are context nodes, by combining contexts through context intersection. Pair (b) shows how context edges that share the same source and destination can be substituted by

[31] Note that the notation for context specifiers that was introduced at the end of the previous chapter is used in examples, while the formal notation continues to be used in definitions.

[32] As it will become evident in the following sections, Multidimensional Data Graphs that are made of "equivalent" multidimensional entities, are *reduced* to identical OEM graphs under every possible world.

a single context edge, using context union. Using the principles exhibited in pairs (a) and (b), it is easy to show that cyclic paths consisting of context edges are not significant and can be ignored, as depicted in pair (c) [33]. Therefore, context c3 in example (c) is ignored, because it does not represent any world under which &9 or &10 hold that is not already represented by c1 or c2, respectively. Pair (d) shows that a conventional entity (represented by context node &12 on the right) can be considered as a multidimensional entity having only one facet that holds under every world. Finally, pair (e) shows that the trivial case of a multidimensional entity without any facets is equivalent to an entity having facets that do not hold under any world.

4.2.3 Multidimensional Data Graphs

In a Multidimensional Data Graph multidimensional entities and conventional entities are interconnected through conventional OEM-like edges, drawn as single thin lines. Those lines are called **entity edges** and define relationships between objects (as explained in Section 4.2.2, the terms **node** and **object** are used interchangeably). Entity edges depart from context nodes and point to context or multidimensional nodes, since a multidimensional node plays the role of a surrogate for its facets.

The existence of two kinds of nodes and two kinds of edges raises the question of which node – edge combinations are meaningful. The following two are the only constraints on the morphology of a Multidimensional Data Graph:

- A context edge cannot start from a context node.
- An entity edge cannot start from a multidimensional node.

A **path** in a Multidimensional Data Graph can consist of context edges and entity edges in any sequence.

As in OEM, context nodes are divided into **complex** and **atomic**. Atomic objects have a value from one of the basic types, e.g. integer, real, strings, etc. The value of a complex object is a set of object references, represented by entity edges. Similarly, the value of a multidimensional object is also a set of object references, represented by context edges. Equivalently, edges may not be treated as incorporated in objects values, but be considered in their own right instead. In this case the values of complex objects and multidimensional objects are assumed to be the reserved values `C` and `M` respectively.

In what follows we formally define Multidimensional Data Graph. As explained in step 4, function *v* uses the reserved values `C` and `M` to classify nodes to multidimensional, (context) complex, and (context) atomic.

Definition 4.3

Let CS be the set of all context specifiers, L be the set of all labels, and A be the set of all atomic values. A ***Multidimensional Data Graph*** *G is a finite directed edge-labeled multigraph* $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$*, where:*

1. *The set of nodes V consists of* ***multidimensional nodes*** *and* ***context nodes****,* $V = V_{mld} \cup V_{cxt}$*. Context nodes are divided into* ***complex nodes*** *and* ***atomic nodes****,* $V_{cxt} = V_c \cup V_a$*.*

[33] This is not to say that context edges formulating a cycle are obsolete, but that the cyclic path itself can be ignored when evaluating a multidimensional entity. In the special case where the cyclic path consists of just one context edge, like in Figure 4.2 (c), the edge is actually obsolete.

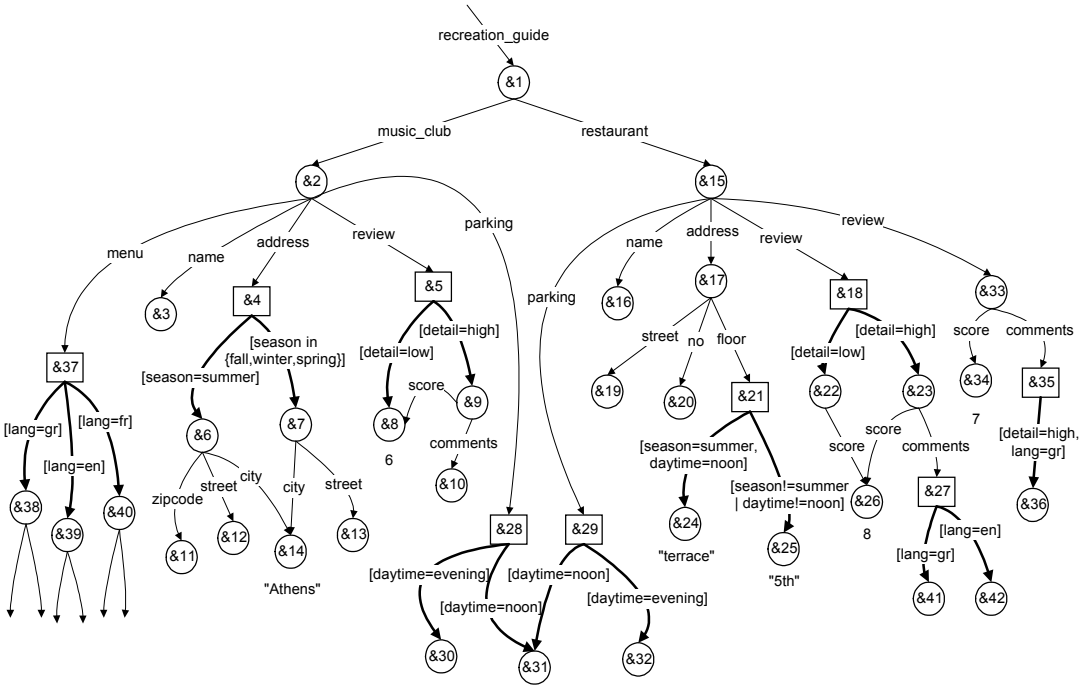
2. *The set of edges E consists of **context edges** and **entity edges**, $E = E_{cxt} \cup E_{ett}$, such that $E_{cxt} \subseteq (V_{mld} \times CS \times V)$ and $E_{ett} \subseteq (V_c \times L \times V)$.*

3. *$r \in V$ is the **root**, with the property that there exists a path from r to every other node in V.*

4. *v is a function that assigns values to nodes, such that: $v(x) =$ `M` if $x \in V_{mld}$, $v(x) =$ `C` if $x \in V_c$, and $v(x)=v'(x)$ if $x \in V_a$, where `M` and `C` are reserved values, and v' is a value function $v': V_a \rightarrow A$ which assigns values to atomic nodes.*

Note that atomic nodes in a Multidimensional Data Graph must be leaves, but leaves are not restricted to atomic nodes and can be any kind of node, even complex or multidimensional. In addition, the root of a Multidimensional Data Graph may be a context node or a multidimensional node. It is easy to recognize that OEM is a special case of Multidimensional Data Graph, where there are no multidimensional nodes and context edges, and where leaves are restricted to atomic nodes.

Figure 4.3: A Multidimensional Data Graph example depicting a context-dependent recreation guide.



The Multidimensional Data Graph in Figure 4.3 is an example of a context-dependent recreation guide[34]. For simplicity, the graph is not fully developed and some of the atomic objects do not have values attached. The dimensions and their respective domains in Figure 4.3 are as follows: `season` ranging over {`summer, fall, winter, spring`}, `daytime`

[34] Like in OEM, we use an additional labeled edge that points to the root in order to give a name to the root, and by extension to the graph and the corresponding semistructured database.

ranging over {noon, evening}, detail ranging over {high, low}, and lang ranging over {en, fr, gr}. The restaurant with oid &15 normally operates on the fifth floor, but at summer noon it operates on the terrace. Therefore, floor with oid &21 is a multidimensional object whose (atomic) value depends on dimensions season and daytime.

Except from having a different value, context objects can have a different structure, as is the case of &6 and &7 which are facets of the multidimensional object address with oid &4. In this case, the music_club with oid &2 operates on a different address during the summer than the rest of the year (in Athens, Greece, it is not unusual for clubs to move south close to the sea in the summer period, and north towards the city center during the rest of the year). The menu of the club is available in three languages, namely English, French and Greek. The restaurant and the club have a number of reviews that can be detailed or brief, depending on the dimension detail. In addition, each has a couple of alternative parking places, depending on the time of day as expressed by the dimension daytime.

4.2.4 Set of Dimensions

Context specifiers in a Multidimensional Data Graph are interpreted with respect to a set of dimensions **D**, which is assumed to accompany the graph. However it is important to realize that, as shown in the previous chapter, **D** is only necessary when transforming a context specifier into a set of worlds through context expansion or context expanded extension; for the rest of context operations, it is *not* necessary to take the actual **D** into account.

For this reason, in what follows we will assume that the set of dimensions **D** for a Multidimensional Data Graph G consists of the dimensions that are encountered in G and none other, unless explicitly stated otherwise.

Dimension domains are needed for all context operations except context intersection, context union, and context extension. For simplicity, the domain of a dimension is assumed to consist of the values of that dimension that are encountered in a Multidimensional Data Graph and none other, unless explicitly stated otherwise.

There is an important difference between those two assumptions. The latter assumption concerning dimension domains is just a convention for simplifying the examples hereafter; in a real application, using the actual domains will probably yield different results. The former assumption concerning the set of dimensions can be seen more as an optimization; it does not really matter what the actual set of dimension is, unless context expansion or context expanded extension needs to be performed.

4.3 CONTEXT PROPAGATION

Context specifiers attached to context edges are the *intended* contexts but not necessarily the *actual* contexts of the respective facets. To understand why, consider a multidimensional entity that evaluates to a facet *f* under a world *w*, and is pointed to by an object that does not hold under *w*. Then *f* does not survive under *w*, because its father does not exist under *w*. This observation leads to the definition of **explicit context** below, and to the notion of **inherited context**, which is the topic of the following section.

Definition 4.4

*Let $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ be a Multidimensional Data Graph. The **explicit context** of an edge $h = (p, k, q)$ is k if $h \in E_{cxt}$, and $\{\varnothing\}$ if $h \in E_{ett}$.*

Therefore, the context specifier attached to a context edge is the explicit context of that edge, while the explicit context of an entity edge is the universal context $\{\varnothing\}$ (also denoted `[]` as we have seen), implying that entity edges are intended to hold under every world. Nodes do not have an explicit context.

In what follows, we define the explicit context of a path.

Definition 4.5

*Let $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ be a Multidimensional Data Graph, and $g_1, g_2, ...g_n$ be edges in E that define a path p in G. Let $ec_1, ec_2, ...ec_n$ be the respective explicit contexts of $g_1, g_2, ...g_n$. Then, the **path explicit context** of p is $ec_p = ec_1 \cap^c ec_2 \cap^c ... \cap^c ec_n$.*

Path explicit context gives the worlds under which the whole path is intended to hold, as prescribed by the explicit contexts of edges in the path.

4.3.1 Inherited Context

The explicit context can be considered as the "true" context only within the boundaries of a single multidimensional entity. When entities are connected together in a multidimensional graph, the explicit context of an edge is not the "true" context, in the sense that it does not alone determine the worlds under which the destination node holds. The reason for this is that, when an entity e_2 is part of (pointed to through an edge) another entity e_1, then e_2 can have substance only under the worlds that e_1 has substance. This can be conceived as if the context under which e_1 holds is inherited to e_2. The context propagated in that way is combined with (constrained by) the explicit context of each edge to give the **inherited context** for that edge. Unlike edges, nodes do not have an explicit context; like edges, however, nodes do have an inherited context.

It is important to emphasize that, from now on[35] we consider inherited contexts instead of explicit contexts for determining the worlds under which a node or an edge holds. In particular, a node or an edge holds under a world w if the inherited context of that node or edge covers w. It follows that a node or an edge holds under a context c if its inherited context is context superset of c.

Definition 4.6

*Let $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ be a Multidimensional Data Graph, ic_r be the inherited context of r, and p be a node in V with $p \neq r$. The **inherited context of node** p is $ic_p = ic_1 \cup^c ic_2 \cup^c ... \cup^c ic_n$, with $n \geq 1$, where $ic_1, ic_2, ..., ic_n$ are the inherited contexts of the edges in E that lead to p. Let q be a node in V, ic_q be the inherited context of node q, h be an edge in E that departs from q, and ec_h be the explicit context of h. The **inherited context of edge** h is a context specifier ic_h, such that: (a) $ic_h = ic_q \cap^c ec_h$ and (b) ic_h represents the least set of worlds.*

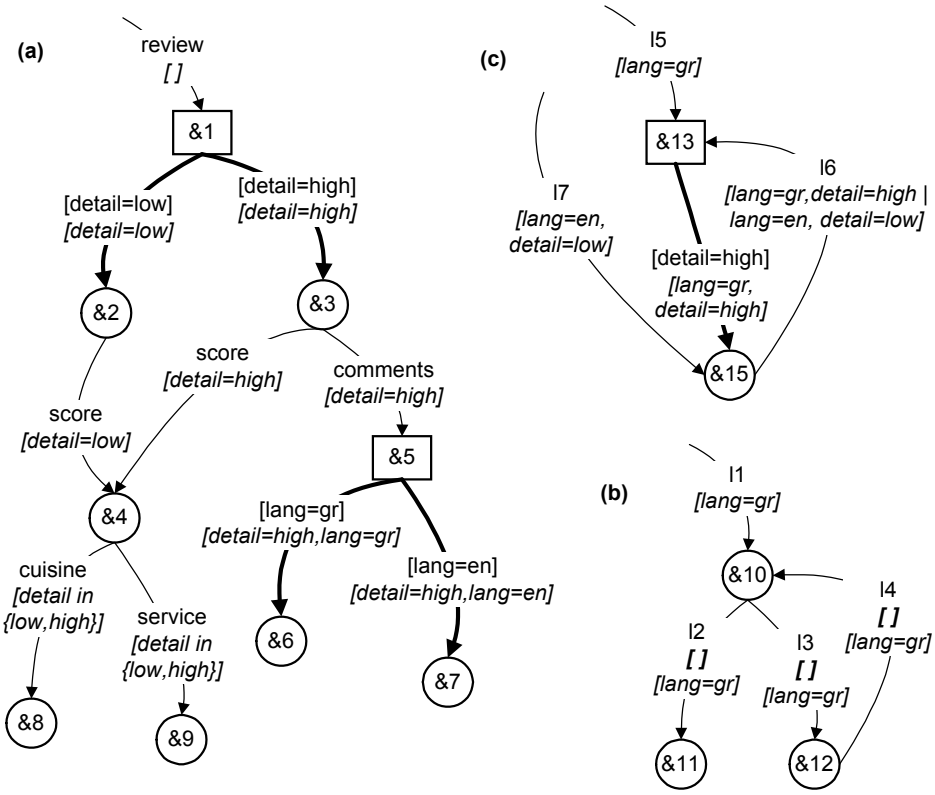
[35] ...from now on and until we introduce **inherited coverage**, which will replace inherited context in determining the worlds under which a node or an edge actually holds.

Condition (b) in the definition of inherited context of an edge is discussed further in the following section.

4.3.1.1 Calculation of Inherited Context

If the inherited context of the root is known, we can calculate the inherited context of every node and edge by traversing the graph in a breadth-first manner. In case the graph does not contain cycles, a simple breadth-first traversal will suffice: for every node that we visit, the inherited context of its parents has already been calculated, thus it is possible to find the inherited context of the current node by calculating the inherited contexts of the incoming edges. This case is depicted in Figure 4.4 (a), which shows a modified part of the graph in Figure 4.3. The inherited context of edges is written in italics, beneath the explicit contexts / labels of context edges / entity edges respectively. The inherited context of a node is the context union of the inherited contexts of incoming edges. For example, the inherited context of node &4 is `[detail=low | detail=high]` which is context equal to `[detail in {low,high}]`.

Figure 4.4: Calculation of inherited context.



Things are more complicated in case the graph contains cycles. The inherited context of a node or edge that constitutes part of a cycle is eventually defined in terms of itself. It will be shown that in such cases a *least fixed point* [Ull88] exists, which gives the inherited context of the node or edge. In Figure 4.4 (b) the inherited context of node &10 depends on that of edge l4, which eventually depends on the inherited context of node &10 again. Suppose we assume that edge l4 has the hypothetical inherited context `[]`, marked in bold in Figure 4.4 (b). Then,

the inherited context of nodes &10, &11, &12 and of edges l2, l3 will also be `[]`, a result that conforms fully to the definition of the inherited context as far as context unions and context intersections are concerned. This result however does not conform to the condition in Definition 4.6 stating that the inherited context of an edge must represent the least set of worlds. The correct inherited context of nodes &10, &11, &12 and of edges l2, l3, l4 is `[lang=gr]`. A similar case is demonstrated in Figure 4.4 (c), where the actual inherited contexts that represent the least set of worlds are written on edges in italics. To find the inherited context of node &13, we start with the initial assumption that it is `[lang=gr]`, like the inherited context of edge l5. Based on that, we calculate the inherited context of the context edge, then of node &15 and then of edge l4. After that, we calculate the inherited context of &13 again, which gives a different result this time: `[lang=gr | lang=en, detail=low]`. Those steps are repeated until no more changes occur in the inherited contexts of nodes and edges in the cycle, and the results are context equal to those of the previous pass.

The process described above can be applied to any strongly connected component[36] in a Multidimensional Data Graph, and it is a naïve way to evaluate a least fixed point. It turns out that there always exists a least fixed point giving the inherited context, because:

(a) Context intersection and context union are *monotone*, as we have stated in the previous chapter. In addition, the composition of monotone operations is itself monotone.

(b) The number of worlds that can be represented by any context specifier is upper-bounded, since the set of dimensions and the dimension domains are assumed finite.

The above conditions ensure [Ull88] that, given a set of context equations defining the inherited contexts in a graph, there always exists a solution that conforms to Definition 4.6. This solution is the least fixed point of the context equations. Consequently, the problem of calculating inherited contexts in a graph with cycles can be rephrased to the evaluation of least fixed points. A naïve evaluation method is to repeat re-calculating the inherited contexts, feeding the results of the one round to the next, until no changes are observed between the results of two successive rounds of calculations.

Inherited context can be defined in an alternative way, using the notion of **path inherited context**.

Definition 4.7

*Let $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ be a Multidimensional Data Graph, and $g_1, g_2, ...g_n$ be edges in E that define a path p in G. Let $ic_1, ic_2, ...ic_n$ be the respective inherited contexts of $g_1, g_2, ...g_n$. Then, the **path inherited context** of p is $ic_p = ic_1 \cap^c ic_2 \cap^c ... \cap^c ic_n$.*

Path inherited context represents the worlds under which all edges in a path hold. Note that, because of Definition 4.6, nodes in the path (including the first and the final nodes) hold under the same worlds as well.

The following proposition shows that the inherited context of a node can be defined in terms of path inherited context.

[36] A *strongly connected component* of a directed graph is a subgraph containing distinct nodes q, u such that there exists a path from q to u and from u to q.

Proposition 4.1

The inherited context of a node q is given by the context union of the path inherited contexts of all paths that start from the root and lead to q.

Proof: First, lets consider a world w covered by the context union of the path inherited contexts. Necessarily, w is covered by the inherited context of some edge pointing to q, therefore the inherited context ic_q of q must cover w as well. Now, lets assume that ic_q covers a world w that is not covered by the context union of the path inherited contexts. This means that the inherited context of some edge g pointing to q covers w, but for every path p that starts from the root and contains g, some edge in p does not cover w with its inherited context. Therefore w may have emerged in g only as the result of a preceded cycle. Remember however that Definition 4.6 does not allow cyclic references to introduce new unsolicited worlds in inherited contexts, by requiring that they represent "the least set of worlds". Consequently, the inherited context of q can only be context equal to the context union of the path inherited contexts of all paths that start from the root and lead to q.

The following proposition shows how explicit contexts of edges can be used instead of inherited contexts to calculate path inherited context.

Proposition 4.2

Let $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ be a Multidimensional Data Graph, q be a node in V with inherited context ic_q, and p be a path in G that starts from q. Let ec_p be the path explicit context of p. Then, the path inherited context ic_p of p is given by: $ic_p = ic_q \cap^c ec_p$.

Proof: Let $g_1, g_2, ...g_n$ be the edges in E that form path p, let $ic_1, ic_2, ...ic_n$ be the respective inherited contexts and $ec_1, ec_2, ...ec_n$ be the respective explicit contexts of those edges. From Definition 4.7 the path inherited context is $c_i = (ic_1 \cap^c ic_2 \cap^c ... \cap^c ic_n)$. From Definition 4.6 we observe that ic_q cannot cover less worlds than ic_1, thus c_i is context equal to $(ic_q \cap^c ic_1 \cap^c ic_2 \cap^c ... \cap^c ic_n)$. From Definition 4.6 we see that the inherited context of an edge cannot cover more worlds than its explicit context. Therefore, c_i is context subset of $c_e = (ic_q \cap^c ec_1 \cap^c ec_2 \cap^c ... \cap^c ec_n)$. Lets assume that c_e represents a world w that is not covered by c_i. Then w is covered by every one of $ic_q, ec_1, ec_2, ...ec_n$, thus w must also be covered by every one of $ic_1, ic_2, ...ic_n$ according to Definition 4.6. Consequently, c_e is context equal to c_i and the claim is proven.

Using Proposition 4.1 and Proposition 4.2 we can express the inherited context of a node or edge in terms of the inherited context of the root and of the explicit contexts of edges, avoiding the recursion in Definition 4.6:

Proposition 4.3

The inherited context of a node is the context union of the path inherited contexts for every path leading from the root to that node, where path inherited context is given by the context intersection of all explicit contexts of edges in the path and of the inherited context of the root. Similarly, for the inherited context of an edge we consider all the paths that start from the root and conclude to that edge.

Proof: The proof is evident from Proposition 4.1, Proposition 4.2, and Definition 4.6.

Note that, if a graph contains cycles an infinite number of paths can be generated, even when the graph is finite. To avoid the generation of infinite number of paths we must pay attention not to traverse the same route multiple times. Thus, the alternative definition of the inherited context given in Proposition 4.3 actually transfers the evaluation of least fixed points from context specifiers and worlds, to paths.

An interesting point is that it is not necessary to take into account the set of dimensions **D** or the dimension domains for calculating the inherited contexts of a Multidimensional Data Graph, because, as shown in the previous chapter, context intersection and context union do not depend on **D** or on the domains of dimensions[37].

4.3.1.2 Significance of Inherited Context

Starting from the root, the nodes and edges of a Multidimensional Data Graph G that hold under a world w formulate a *subgraph of G that holds under w*[38]. A Multidimensional Data Graph G may consolidate a number of graphs holding under various contexts, as different subgraphs of G may have substance under different worlds.

The inherited context of a node or edge represents the worlds under which the node or edge "survives" as a part of such subgraphs. The reason is that the inherited context is actually a propagation of constraints from the root to the leaves, consequently if the inherited context of a node or edge covers a world w, then no constraint prevents the node or edge to hold under w [39].

Given a Multidimensional Data Graph G, the subgraphs of G that hold under various possible worlds will themselves be Multidimensional Data Graphs and will have the same root as G. Here is why: Because of the definition of inherited context, if the inherited context of an edge covers a world w, then w is also covered by the inherited contexts of the two nodes this edge connects. Therefore, if an edge holds under a world w, then its departure and destination nodes hold under w as well; in other words, surviving edges always point to some surviving node. In addition, an important conclusion stemming from Proposition 4.1 is that, if the inherited context of a node or edge covers a world w, then that node or edge must be part of some path that starts from the root and whose path inherited context covers w as well. In other words, the nodes and edges of a Multidimensional Data Graph that hold under a world w *are always accessible from the root through some path that holds under w*.

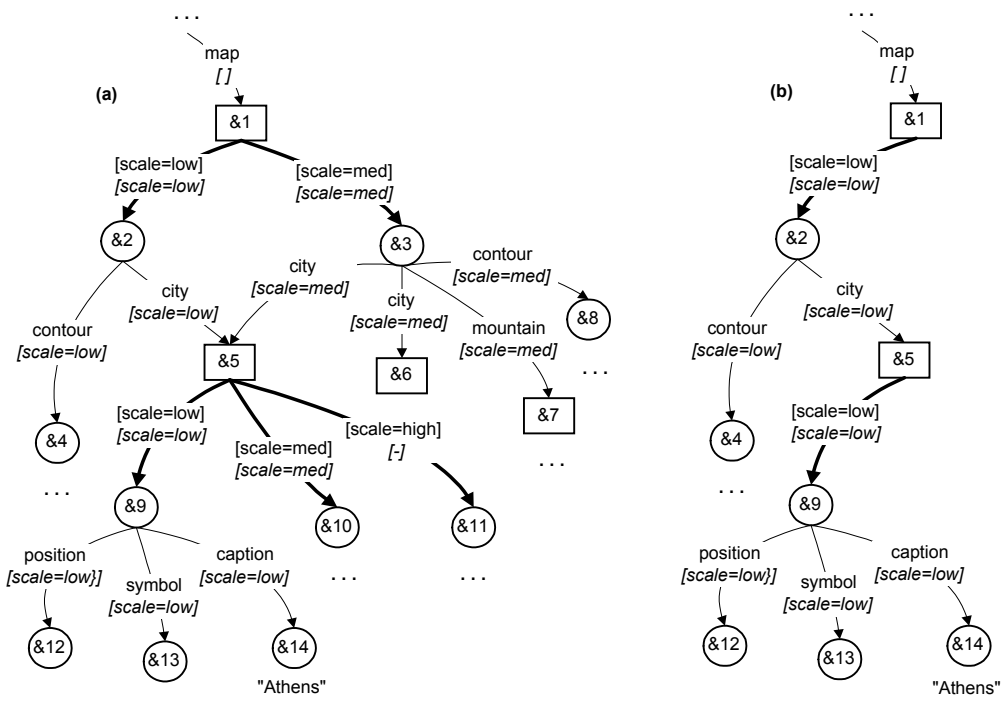
---

[37] In the example of Figure 4.4 (c) we used context equality, which depends on dimension domains. However, as shown by Proposition 4.3, even when there are circles as in the case of the example in Figure 4.4 (c), it is possible to calculate the inherited contexts without using context equality. Dimension domains are necessary however, if context specifiers in the graph contain syntax elements like "`!=`" or "`not in`".

[38] Unless explicitly stated otherwise, we always refer to the *maximum* subgraph that holds under w.

[39] This is not entirely true, as we will see in Section 4.3.2, where we introduce **context coverage**.

Figure 4.5: The subgraph of (a) that holds under the world [scale=low] is shown in (b).



As a simple example, consider the graph depicted in Figure 4.5 (a) that represents cartographic information for a map that can be displayed in a number of different *scales* (higher scales means that more features are shown on the map). When scale is low, the map (oid &2) consists of a rough contour (oid &4) of the area, and the most significant city (oid &5). When scale is medium, the map (oid &3) comprises a more detailed contour (oid &8), a mountain (oid &7), and a couple of cities (oids &5 and &6). Map features may in turn have different facets depending on the scale, which determine the presentation detail and characteristics of those features. For example, the city with oid &5 has three facets, for low (oid &9), medium (oid &10), and high (oid &11) scale. The low scale facet contains only a small symbol (oid &13) marking the city on the map, the position (oid &12) of the symbol, and the name (oid &14) of the city, as it will appear on the map. The high scale facet may be a detailed map of the city itself, containing information about the blueprint of streets and buildings.

The inherited context of edges in Figure 4.5 is given in the second line of edge labels, and is written in italics. Using the inherited context of nodes and edges it is easy to see whether a node or edge holds under some world. For example, assuming the world w = `[scale=low]`, the node with oid &5 holds under w because its inherited context is `[scale in {low, med}]` and covers w. The subgraph of Figure 4.5 (a) that holds under w is depicted in Figure 4.5 (b).

In Figure 4.5 (a) notice that the edge *(&5, [scale=high], &11)* and the node &11 (which is the high scale facet of the city with oid &5) have as inherited context the empty context. The parts of a graph whose inherited context is the empty context do not hold under any world and are not contained in any subgraph holding under some world. It must be emphasized that *such parts are still important*, as they may contain useful information. For example, consider

the following query on the graph of Figure 4.5 (a): "give me in the highest detail a plan of the city whose caption on the low scale map is Athens". The answer to this query would involve node &11, whose inherited context is the empty context.

4.3.1.3 Inherited Context of Root

From the definition of inherited context, it follows that the inherited context of the root poses a restriction to the whole graph. Intuitively, by constraining the inherited context of the root, one can "project" the whole graph: if the root does not hold under some world, then none of its descendants may hold under that world. Consequently, given a Multidimensional Data Graph G, no subgraph of G may hold under a world that is not covered by the inherited context of the root. It is easy to see that if we restrict the inherited context of the root, then the inherited contexts of all nodes and edges in the graph will be respectively restricted.

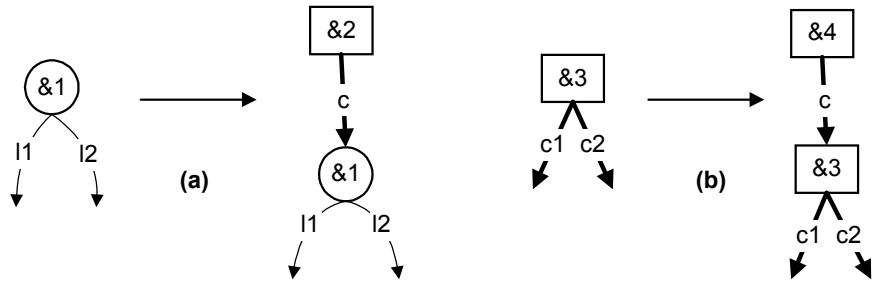
Proposition 4.4

If the inherited context of the root is restricted from ic_r to $ic_r' \subset^c ic_r$, then the new inherited context ic_{new} of any node or edge is given by $ic_{new} = ic_{old} \cap^c ic_r'$, where ic_{old} is the previous inherited context of that node or edge.

Proof: From Proposition 4.3 the inherited context ic of any node or edge is given by $p_1 \cup^c p_2 \cup^c ... \cup^c p_k$, where p_i is the path inherited context of a path i starting from the root and concluding to that node or edge. If we denote g_i the context intersection of the explicit contexts of edges in path i, then ic becomes $(ic_r \cap^c g_1) \cup^c (ic_r \cap^c g_2) \cup^c ... \cup^c (ic_r \cap^c g_k)$, where ic_r is the inherited context of the root. Because of the distributive law, ic is given by $ic_r \cap^c (g_1 \cup^c g_2 \cup^c ... \cup^c g_k)$. Thus, since $ic_r' =^c ic_r \cap^c ic_r'$, restricting the root to ic_r' entails the context intersection of ic with ic_r'.

If the root holds under every possible world, then the inherited context of the root is the universal context. It is convenient to assume that the inherited context of the root is always the universal context. Figure 4.6 shows how we can restrict the inherited contexts in a graph, while maintaining the universal context as the inherited context of the root.

Figure 4.6: Restricting the inherited context of the root to c.



In Figure 4.6 (a) the inherited context of the root with oid &1 is `[]`, and we want to change it to c. A new multidimensional node with oid &2 is added, that points to &1 through a context edge having c as its explicit context. The new node becomes the new root, with inherited context `[]`. The inherited context of the old root (oid &1) is now c. Figure 4.6 (b) demonstrates the same method in case the initial root is a multidimensional node.

From this point on, we will assume that the inherited context of the root of a Multidimensional Data Graph is the universal context `[]`, unless explicitly stated otherwise.

4.3.2 Context Coverage

As explained in the previous section, a Multidimensional Data Graph may comprise a number of subgraphs, which are themselves Multidimensional Data Graphs, each holding under some world(s). At this point we may remember that leaves in a Multidimensional Data Graph can be any kind of nodes, atomic, complex, or multidimensional[40], while leaves in OEM can only be atomic nodes. In both kinds of graphs, atomic nodes are the ones that actually bare data, therefore the question that arises is: *under what conditions are the leaves of a subgraph that holds under some world atomic nodes?*

In order to properly define a relationship between Multidimensional Data Graph and OEM, we need some way to determine whether a node in a Multidimensional Data Graph reaches to atomic nodes under some world. This is achieved by introducing the notion of **context coverage**. The context coverage of nodes and edges is given by the propagation of explicit contexts from the leaves upwards to the root of the graph.

Definition 4.8

*Let $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ be a Multidimensional Data Graph, and p be a node in V that is a leaf. The **context coverage of leaf node** p is the context specifier $\{\{-\}\}$ if $p \in V_{mld} \cup V_c$, and $\{\varnothing\}$ if $p \in V_a$. Let q be a node in V that is not a leaf. The **context coverage of internal node** q is $cv_q = cv_1 \cup^f cv_2 \cup^f ... \cup^f cv_n$, with $n \geq 1$, where $cv_1, cv_2, ..., cv_n$ give the context coverage of the edges in E that depart from q. Let u be a node in V, cv_u be the context coverage of u, h be an edge in E that points to u, and ec_h be the explicit context of h. The **context coverage of edge** h is a context specifier cv_h, such that: (a) $cv_h = cv_u \cap^c ec_h$ and (b) cv_h represents the least set of worlds.*

Context coverage is similar to inherited context, with explicit contexts propagating towards the opposite direction.

4.3.2.1 Calculation of Context Coverage

Calculating context coverage is essentially the same as the calculation of inherited context, discussed in Section 4.3.1.1. The following propositions about context coverage correspond to those in Section 4.3.1.1 about inherited context.

Definition 4.9

*Let $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ be a Multidimensional Data Graph, and $g_1, g_2, ...g_n$ be edges in E that define a path p in G. Let $cv_1, cv_2, ...cv_n$ be the respective context coverages of $g_1, g_2, ...g_n$. Then, the **path context coverage** of p is $cv_p = cv_1 \cap^c cv_2 \cap^c ... \cap^c cv_n$.*

[40] Even if we required that Multidimensional Data Graph have atomic nodes as leaves, nothing guarantees that those leaves would be accessible under any world. Our approach is to introduce Multidimensional Data Graph as a general, cover-all structure, and then to define MOEM as a more restricted graph that exhibits additional properties.

Proposition 4.5

The context coverage of a node q is given by the context union of the path context coverages of all paths that start from q and lead to a leaf.

Proof: Similar to Proposition 4.1.

Proposition 4.5 can be equivalently restated by considering paths leading to atomic nodes only, since paths leading to leaves that are complex or multidimensional nodes do not contribute any world to the context coverage of q.

Proposition 4.6

Let $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ be a Multidimensional Data Graph, q be a node in V with context coverage cv_q, and p be a path in G leading to q. Let ec_p be the path explicit context of p. Then, the path context coverage cv_p of p is given by: $cv_p = cv_q \cap^c ec_p$.

Proof: Similar to Proposition 4.2.

Proposition 4.7

The context coverage of a node is the context union of the path context coverages for every path leading from that node to some atomic node, where path context coverage is given by the context intersection of all explicit contexts of edges in the path. Similarly, for the context coverage of an edge we consider all the paths that start with that edge and end with some atomic node.

Proof: The proof is evident from Proposition 4.5, Proposition 4.6, and Definition 4.8. In addition, the following simplification is assumed: since the context coverage of an atomic node is the universal context, it is not necessary to take it into account when using Proposition 4.6 to calculate the path context coverage of a path leading to that atomic node.

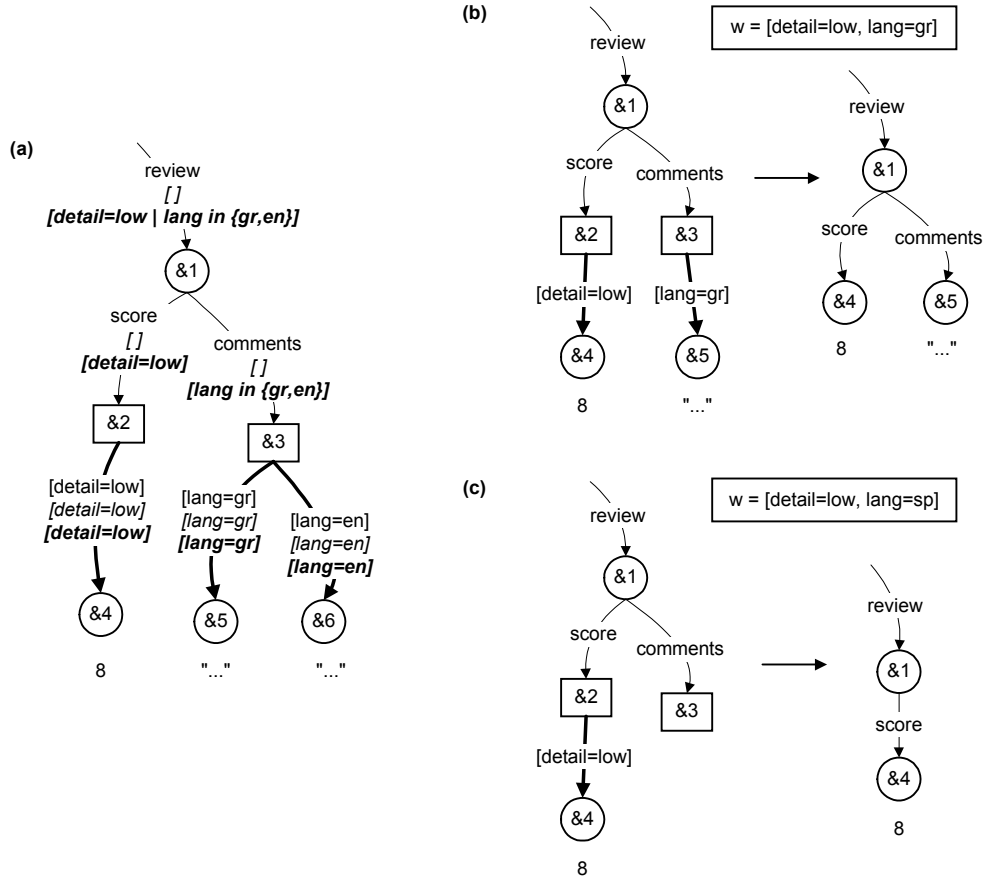
Evidently, the techniques for calculating inherited context can be employed for the calculation of context coverage as well. A simple way to apply this principle is to change the direction of every edge in a graph, and calculate the inherited context for every subgraph whose root is an atomic node of the initial graph. Because of Proposition 4.1 and Proposition 4.5, the context union of all inherited contexts (one for every atomic node) for a node or edge will give the context coverage of that node or edge.

4.3.2.2 Significance of Context Coverage

The context coverage of leaves that are complex nodes or multidimensional nodes is defined to be the empty context {{–}} (also denoted `[-]`), while the context coverage of leaves that are atomic nodes is defined to be the universal context {∅} (also denoted `[]`). Stating the obvious, this encodes the rule that atomic nodes can be leaves under any possible world, while complex and multidimensional nodes cannot be leaves under any world.

Proposition 4.7 shows that *the context coverage of an internal node q represents the worlds under which there exists a path from q to some atomic node*. To clarify why this is important, the following example gives an intuition of cases where it is possible to "extract" from a Multidimensional Data Graph a conventional OEM holding under a particular world.

Figure 4.7: Context coverage and its relation to extraction of conventional OEMs.



In Figure 4.7 (a) a part of an OEM representing a restaurant review is shown. The `review` object (oid &1) is a complex object consisting of two multidimensional subobjects, namely `score` (oid &2) and `comments` (oid &3). Object `score` has only one facet (oid &4) holding under low detail, while `comments` has two facets (oid &5 and oid &6) holding under the worlds where language is Greek and English, respectively. The inherited contexts of edges are written in italics on the second line of labels, and their context coverages are written in bold italics on the third line of labels.

The Multidimensional Data Graph at the left side of Figure 4.7 (b) is the subgraph of (a) that holds under the world w = `[detail=low, lang=gr]`, as discussed in Section 4.3.1.2: nodes and edges of graph in (a) whose inherited context covers w constitute the subgraph in (b). It is easy to see that from this subgraph we can omit the multidimensional nodes and the context edges and get a conventional OEM holding under w, which is depicted at the right side of Figure 4.7 (b). A similar case occurs when we consider the world where detail is low again, and language is English instead of Greek. However, if Spanish is one of the possible languages and we consider the world w′ = `[detail=high, lang=sp]`, it is not possible to extract an OEM that holds under w′. The reason is that all the leaves of the Multidimensional Data (sub-) Graph holding under w′ are multidimensional nodes, and if those nodes are omitted in the process of extracting an OEM the complex `review` object with oid &1 will be left without children. At this point, notice that the context coverage of the `review` object in

the initial graph of Figure 4.7 (a) contains the worlds under which it is possible to extract an OEM, but does not contain the world w′, under which it is not possible to extract an OEM.

An interesting case is shown in Figure 4.7 (c). Under the world w = `[detail=low, lang=sp]` the leaves of the graph on the left are mixed: one is an atomic node (oid &4) while the other is a multidimensional node (oid &3) [41]. There are two ways to deal with this situation:

(a) *Strict typing approach*: Requires that the `review` object have the same structure (type) under every possible world. Under this condition the graph on the left in Figure 4.7 (c) cannot give an OEM.

(b) *Varying structure approach*: Relaxes strict type constraints, and allows the `review` object to have different structure under different worlds. In this case it is possible to extract an OEM that holds under w, which is shown on the right side of Figure 4.7 (c).

The approach we adopted is the varying structure approach, because it is in line with the varying structure exhibited by semistructured data. Remember that facets of the same multidimensional entity may have different structure; therefore a multidimensional object can already have different types under different worlds. In addition to that, the approach we adopted relates the structure of an object under some world to its multidimensional subobjects, and provides an alternative, more economical[42] way to vary structure under different worlds. The worlds under which a context node u cannot include in its structure a child node q because of inaccessibility of q to atomic nodes, are given by $ic_u -^c cv_q$, where ic_u is the inherited context of u, and cv_q is the context coverage of q.

Although the strict typing approach is probably not best suited for MSSD, it is certainly a valid approach, and the most natural to follow for (completely) structured data. To conform to the strict typing approach, Definition 4.8 of context coverage should be modified as follows: the context coverage cv_q of an *internal complex* node q should be given by $cv_1 \cap^c cv_2 \cap^c ... \cap^c cv_n$ instead of $cv_1 \cup^c cv_2 \cup^c ... \cup^c cv_n$, therefore involving the context intersection and not the context union of the context coverage of outgoing entity edges. The strict typing approach deserves more attention, however in the frame of MSSD we will focus on the varying structure approach.

As a conclusion, *the context coverage of a context node or an entity edge represents the set of worlds under which this node or edge is allowed to participate in extracted OEMs, as imposed by accessibility to atomic nodes*.

4.3.2.3 Context Coverage of Root

The following proposition gives the context coverage of the root directly from the inherited context of atomic nodes.

---

[41] It is possible to have complex nodes as leaves in a subgraph holding under some world, but only if those complex nodes also appear as leaves in the initial Multidimensional Data Graph. Because of the definition of inherited context, a complex node that is internal in the initial Multidimensional Data Graph cannot appear as a leaf in a subgraph holding under some world. Complex nodes that appear as leaves in subgraphs holding under a world are treated in the same way as multidimensional nodes that appear as leaves (for example, see node &3 in Figure 4.7 (c)).

[42] In the graph of Figure 4.7 (a), instead of changing the structure of `review` through its multidimensional subobjects, we could have made `review` a multidimensional object with facets that had different structures and that pointed directly to the proper context nodes.

Proposition 4.8

Let G be a Multidimensional Data Graph. The context coverage of the root of G is given by the context union of the inherited contexts of the atomic nodes in G.

Proof: Obvious, from Proposition 4.1, Proposition 4.2, Proposition 4.5, and Proposition 4.6.

For a world w covered by the context coverage of the root, there exist atomic nodes that are accessible under w, and it is possible to extract an OEM holding under w. On the other hand, if w is *not* contained in the context coverage of the root, the subgraph that holds under w cannot give a proper OEM, since the root will not have access to any atomic node. Therefore, *the context coverage of the root of a Multidimensional Data Graph G represents the worlds under which it is possible to extract OEMs from G*.

An assumption implied by the above statement is that, as explained in Section 4.3.1.3, the inherited context of the root is the universal context. If this is not the case, the worlds under which it is possible to extract OEMs are given by $ic_r \cap^c cv_r$, where ic_r is the inherited context of the root and cv_r is the context coverage of the root. This expression, as we shall see in the following section, is called **inherited coverage** and applies not only to the root, but also to every node and edge in a Multidimensional Data Graph.

4.3.3 Inherited Coverage

As we have seen, the inherited context of a node or an edge expresses accumulated constraints as imposed by ancestors in the graph, while context coverage expresses accumulated constraints as imposed by descendants. The context intersection of the two gives the **inherited coverage** of a node or an edge.

Definition 4.10

*The **inherited coverage** of a node or edge in a Multidimensional Data Graph is $icv = ic \cap^c cv$, where ic is the inherited context and cv is the context coverage of that node or edge.*

Therefore, *the inherited coverage of a node or an edge represents the worlds under which it is meaningful for the node or edge to exist. Under each such world, there holds a path that starts from the root, terminates at some atomic node, and comprises that node or edge*.

Figure 4.8: A graph annotated with the inherited coverage of edges.

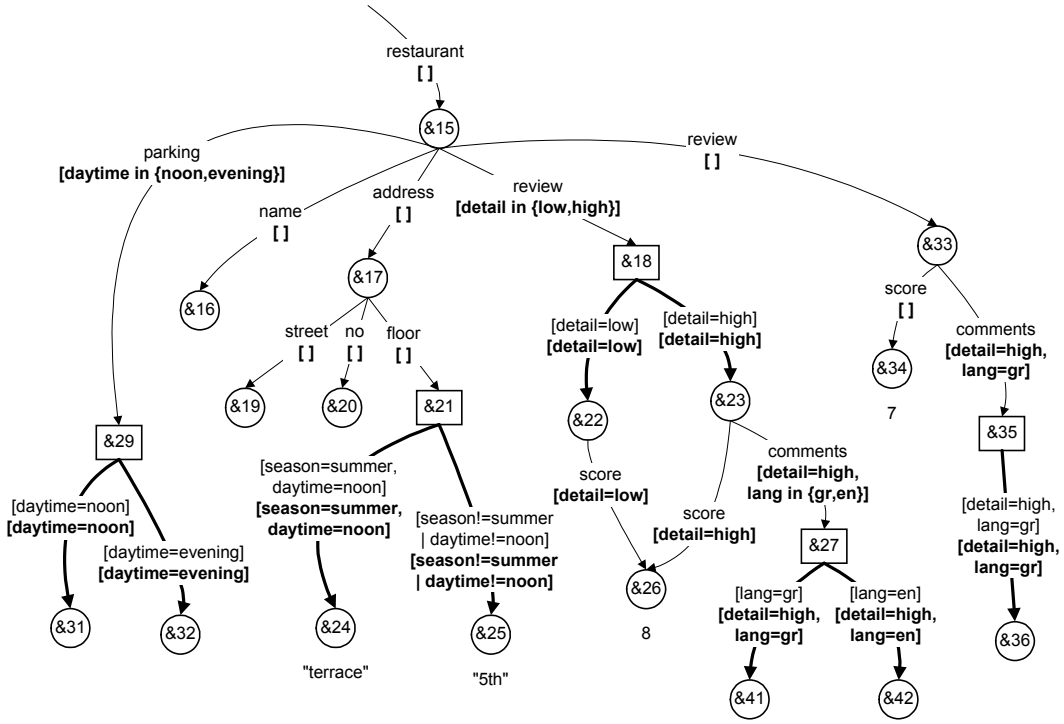


Figure 4.8 shows a part of the Multidimensional Data Graph depicted in Figure 4.3 annotated with the inherited coverage of edges, which is written in bold beneath the normal edge label. Note that all leaves in Figure 4.8 are considered atomic nodes, although some of their values have been omitted for simplicity.

The accessibility to some atomic node is a necessary prerequisite for a node or an edge to have meaning under some world. Therefore, *in what follows we will consider inherited coverage instead of inherited context for determining the worlds under which a node or an edge actually holds*. In particular, a node or an edge holds under a world w if the inherited coverage of that node or edge covers w. It follows that a node or an edge holds under a context c if its inherited coverage is context superset of c.

Note that the inherited coverage is context subset of inherited context, therefore the worlds under which a node or an edge holds with respect to its inherited coverage are a subset of the worlds under which that node or edge holds with respect to its inherited context.

4.3.3.1 Path Inherited Coverage

In analogy to path inherited context and path context coverage, in what follows we introduce **path inherited coverage**.

Definition 4.11

*The **path inherited coverage** of a path p in a Multidimensional Data Graph is* $icv_p = ic_p \cap^c cv_p$*, where* ic_p *is the path inherited context of p and* cv_p *is the path context coverage of p.*

The following propositions give alternative ways to calculate path inherited coverage.

Proposition 4.9

Let $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ be a Multidimensional Data Graph, and $g_1, g_2, ...g_n$ be edges in E that define a path p in G. Let $icv_1, icv_2, ..., icv_n$ be the respective inherited coverages of $g_1, g_2, ...g_n$. Then, the path inherited coverage of p is given by $icv_p = icv_1 \cap^c icv_2 \cap^c ... \cap^c icv_n$.

Proof: Obvious, from Definition 4.7, Definition 4.9, and Definition 4.10.

Proposition 4.10

Let $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ be a Multidimensional Data Graph, q and u be nodes in V, and p be a path in G that starts from q and concludes to u. Let ic_q be the inherited context of q, cv_u be the context coverage of u, and ec_p be the path explicit context of p. Then, the path inherited coverage of p is given by $icv_p = ic_q \cap^c cv_u \cap^c ec_p$.

Proof: Obvious, from Proposition 4.2 and Proposition 4.6.

Notice that given a path p with path inherited coverage icv_p, because of Definition 4.6 and Definition 4.8, the inherited coverage of every node in p (including the first and the final nodes) is context superset of icv_p. Therefore, *path inherited coverage represents the worlds under which all nodes and edges in the path do hold*.

4.3.3.2 Significance of Inherited Coverage

As shown in Section 4.3.1.2, the nodes and edges of a Multidimensional Data Graph G that hold under a world w with respect to their inherited contexts, formulate a subgraph of G with the same root as G that holds under w. Evidently, this subgraph is further pruned if we take the nodes and edges that *actually* hold under w (with respect to their inherited coverages). Lets discuss the properties of this pruned subgraph g.

(a) If a node q belongs to g, then the root r of G belongs to g as well. In addition, there exists a path in g leading from r to q. Here is why: If q belongs to g, then both its inherited context ic_q and its context coverage cv_q contains w. From Proposition 4.3, since ic_q contains w, there exists a path p in G from the root to q such that $ic_r \cap^c ec_1 \cap^c ec_2 \cap^c ... \cap^c ec_n$ contains w, where $ec_1, ec_2, ..., ec_n$ are the explicit contexts of edges in the path. Then, w is also contained in $cv_q \cap^c ic_r \cap^c ec_1 \cap^c ec_2 \cap^c ... \cap^c ec_n$, which according to Proposition 4.10 is the path inherited coverage of p. According to Proposition 4.9, the inherited coverage of every edge in p contains w, therefore every edge and node in p is part of g. It follows that the root r of G is the root of g as well.

(b) If an edge *(q, l, u)* belongs to g, then the nodes q and u will also belong to g. This emanates from Definition 4.8 of context coverage.

(c) Leaves in g are exclusively atomic nodes. In particular, they are the atomic nodes of G whose inherited context contains w. To understand why, consider that multidimensional nodes and complex nodes that are leaves in G cannot be part of g, since their context coverages do not contain any world. Lets assume that a multidimensional node q that is not a leaf in G appears as a leaf in g. Since q is part of g, the inherited context of q contains w, therefore the inherited contexts of the edges departing from q also contain w. Consequently, the reason why the edges departing from q are not part of g is that their context coverages do not contain w. In this case however, the context coverage of q would not contain w either, and q

would not be part of g. For the same reasons, a complex node that is internal in G cannot appear as a leaf in g.

Summarizing, the components of a Multidimensional Data Graph G that hold under a world w (with respect to their inherited coverage), formulate a subgraph, which is itself a Multidimensional Data Graph having the same root as G and whose leaves are exclusively atomic nodes[43]. In Section 4.4.1, where we discuss the process of **reduction to OEM**, we will see that it is always possible to extract a conventional OEM holding under w from such a subgraph.

It is easy to see that the above do not generalize for contexts that represent more than a single world. In particular, the properties (a) and (c) shown above for a single world do not hold when we consider contexts representing more than one world. The case of reduction to OEM under more than one world is discussed in detail in Section 4.4.1.3.

The meaning of inherited coverage is now clear. *The inherited coverage of a node or an edge in a Multidimensional Data Graph G represents the worlds under which the node or edge is taken into account when extracting conventional OEMs from G*. In particular, the inherited coverage of a context node gives the worlds under which the node is part of a conventional OEM. Here is why: a node q that does not have access to any atomic node under a world w cannot be part of an OEM holding under w. Therefore, only worlds contained in its inherited coverage can be considered. In addition, as shown in item (a) above, for any such world w, there exists a path from the root to q that holds under w, and the inherited coverage of the root contains w as well. Similarly, the inherited coverage of an entity edge gives the worlds under which the edge is part of a conventional OEM.

4.3.3.3 Graph Context Projection

In Proposition 4.4 we have shown that if we restrict the inherited context of the root, then the inherited contexts of all nodes and edges in the graph are respectively restricted. The context coverages of nodes and edges in the graph are not affected in this case, whereas the inherited coverages of nodes and edges are respectively restricted, just like inherited contexts.

Proposition 4.11

If the inherited context of the root is restricted from ic_r to $ic_r' \subset^c ic_r$, then the new inherited coverage icv_{new} of any node or edge is given by $icv_{new} = icv_{old} \cap^c ic_r'$, where icv_{old} is the previous inherited coverage of that node or edge.

Proof: Obvious, from Definition 4.10 and Proposition 4.4.

Therefore, by restricting the inherited context of the root as depicted in Figure 4.6 we actually restrict the worlds under which nodes and edges hold, and for which OEMs can be extracted. Thus, we *project the graph* to the worlds represented by the new inherited context of the root. Consequently, **graph context projection** of a Multidimensional Data Graph to a context c involves two steps:

(a) Restriction of the inherited context of the (original) root to c, as shown in Figure 4.6.

(b) Adjustment of the inherited coverages of nodes and edges in the graph, as dictated by Proposition 4.11.

[43] This last property makes all the difference from the subgraphs we have seen in Section 4.3.1.2, which are formulated based on inherited context instead of inherited coverage.

Graph context projection is similar to the process of **partial reduction**, which we discuss in Section 4.4.2. Partial reduction goes further and eliminates nodes and edges that do not hold under any world in the projected graph.

4.4 REDUCTION

It is evident that a Multidimensional Data Graph may comprise a number of conventional OEMs holding under different worlds. In previous sections we discussed the conditions that are *necessary* and *sufficient* in order to be able to extract an OEM holding under some world. In particular, we showed that we can extract OEMs only for the worlds represented by the inherited coverage of the root. In addition, we showed that the inherited coverage of a context node or entity edge represents the worlds under which that node or edge is part of an OEM incorporated in the graph.

Reduction is a process that eliminates some of the nodes and edges of a Multidimensional Data Graph, resulting in a subgraph whose elements satisfy some criteria. Those criteria have to do with inherited coverage, therefore when we refer to a Multidimensional Data Graph we assume that the inherited coverages of its nodes and edges have been calculated and are known. Inherited coverages need to be re-calculated only when the graph is changed, while reductions may occur at any moment, after a user request for example. In this section, we specify an algorithm that performs **reduction to OEM**, holding under a given world, of a Multidimensional Data Graph. We also explain **partial reduction**, which reduces a Multidimensional Data Graph to a new Multidimensional Data sub-Graph with interesting properties. We then show that both types of reduction can be decomposed to the same basic operations.

4.4.1 Reduction to OEM

For reducing a Multidimensional Data Graph G to an OEM holding under some world, it is important to know whether G is a **context-deterministic graph**. The next section defines this concept; the section that follows presents **reduction to OEM** under some world, while the final section discusses OEMs that hold under more than one worlds.

4.4.1.1 Context-Deterministic Graphs

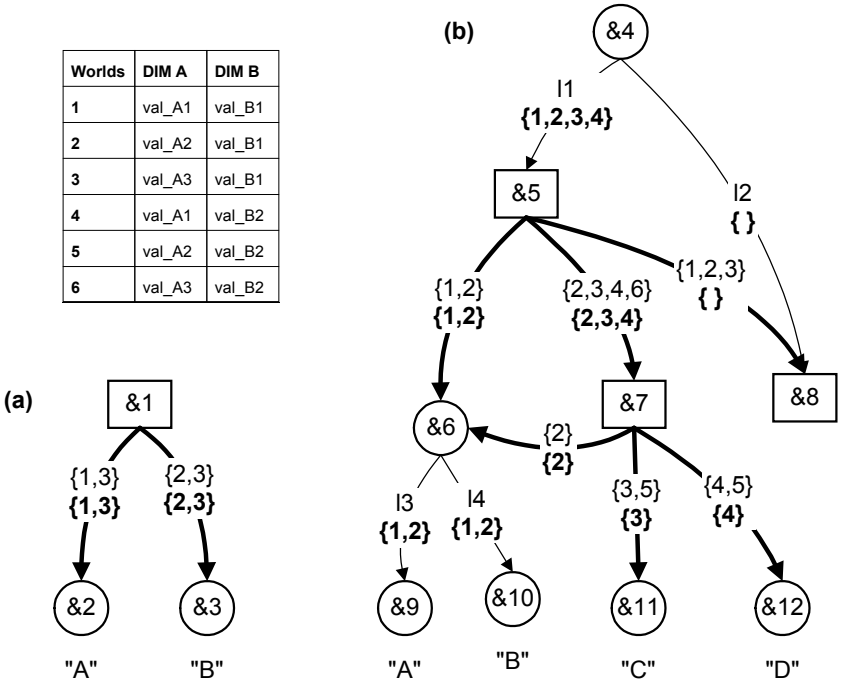
As we have seen, it is possible for a multidimensional entity to encompass more than one facets holding under the same world. The notion of context-determinism imposes some restrictions on this issue. In particular, in context-deterministic Multidimensional Data Graphs facets of the same multidimensional entity that are conventional information entities (in other words, context nodes) must be accessible under disjoint sets of worlds. This, however, applies directly only to some of the multidimensional entities in the graph.

Definition 4.12

*Let $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ be a Multidimensional Data Graph. We say that G is **context-deterministic** iff for every world w and multidimensional node q, which is either the root node or it is pointed to by an entity edge, the following holds: if $p_1, p_2, \ldots, p_n$ are all the paths that start from q, terminate to context nodes, and consist solely of context edges whose inherited coverage contains w, then all those paths $p_1, p_2, \ldots, p_n$ terminate to the same context node.*

Obviously, if a context c represents two or more worlds and p_k, p_j hold under c and lead to different context nodes, the graph is not context deterministic. Therefore, in a context-deterministic graph, given a context c that represents at least one world, if we start from a multidimensional node that is pointed to by an entity edge, we can navigate to at most one context node through consecutive context edges that hold under c. In other words, *in a context-deterministic graph any entity edge pointing to a multidimensional node can be associated with at most one context node under any non-empty context c*.

Figure 4.9: Context-deterministic and context-nondeterministic graphs.



Consider the example in Figure 4.9. In order to simplify the graphs and make them more intuitive, context specifiers in Figure 4.9 appear as the sets of worlds they represent. The correspondence is given at the table on the top left corner, where we assume two dimensions `A` and `B` with domains `{val_A1, val_A2, val_A3}` and `{val_B1, val_B2}` respectively. For every edge its label or its explicit context appears on the first line, while the second line shows the inherited coverage in bold. The Multidimensional Data Graph in Figure 4.9 (a) is context-nondeterministic, because the multidimensional node &1 is the root and is connected to more than one context nodes through context edges that hold under the same world, in particular world `3`. On the other hand, the graph in Figure 4.9 (b) is context-deterministic. Here is why. There are two multidimensional nodes of interest, node &5 and &8. Node &7 is irrelevant, because it is neither the root, nor is it pointed to by an entity edge. Node &8 is a leaf, so it is not connected to any context node. Node &5 is connected to context nodes &6, &11, and &12 through paths that consist exclusively of context edges. Paths `<(&5,{1,2},&6)>`, `<(&5,{2,3,4,6},&7).(&7,{3,5},&11)>`, and `<(&5,{2,3,4,6},&7).(&7,{4,5},&12)>` hold under disjoint sets of worlds, or equivalently, their path inherited coverages (`{1,2}`, `{3}`, and `{4}` respectively) are mutually exclusive. Therefore, given any world w, at most one of those paths may hold under w. The only other path `<(&5,{2,3,4,6},&7).(&7,{2},&6)>` holds under `{2}` together with the

path `<(&5,{1,2},&6)>` which holds under `{1,2}`. Both paths however lead to the same context node (oid &6), so the graph is context-deterministic.

An alternative to Definition 4.12 is that a graph is context-deterministic iff for every multidimensional node q that is either the root or is pointed to by an entity edge, the following is true: let icv_p, icv_u, ..., icv_n be the respective context unions of path inherited coverages of all paths of consecutive context edges that lead from q to context nodes p, u, ..., n; then icv_p, icv_u, ..., icv_n are mutually exclusive with one another.

In Section 4.5 we will see that the **canonical form** of a Multidimensional Data Graph allows one to check whether a graph is context-deterministic in a simpler and more intuitive way.

4.4.1.2 Reduction to OEM

Given a context-deterministic Multidimensional Data Graph $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ and a context specifier w that represents a single world[44], the following process extracts from G a conventional OEM O_w that holds under w.

$O_w \leftarrow$ reduceToOEM (G, w) is:

1. Remove every node from V and every edge from E whose inherited coverage is not context superset of w.
2. For every $(q, l, p) \in E_{ett}$ such that $p \in V_{mld}$, remove (q, l, p) from E_{ett}, and add (q, l, u) to E_{ett} if it does not already exist, where u is a context node such that a path of context edges leads from p to u. In addition, if the root r is a node $p \in V_{mld}$, then set r to node u, where u is a context node such that a path of context edges leads from p to u.
3. Return $O_w = (V_{cxt}, E_{ett}, r, v)$.

The OEM O_w is then the facet of the Multidimensional Data Graph G under w.

Lets discuss the process **reduceToOEM** in more detail. As we have seen in Section 4.3.3.2, after Step 1 of the process the remaining nodes and edges of G have the following properties: (a) they all hold under w, (b) they form a subgraph which is also a Multidimensional Data Graph and has the same root as G, and (c) leaves are exclusively atomic nodes. In Section 4.3.3.2 we claimed that it is always possible to extract an OEM from such a subgraph. The OEM construction is effectively performed in Step 2 of the process. Step 2 takes every entity edge that points to a multidimensional node p and makes it point to a context node u. As we have seen in Section 4.4.1.1, node u is the only facet of p accessible from p under w, because G is context-deterministic. In addition, node u must exist; otherwise p would not have access to some atomic node under w, and would have been eliminated in Step 1. Consequently, every entity edge pointing to a multidimensional node can be properly redirected to point to the correct context node, substituting a path of context edges. Note that if an edge identical to the redirected one already exists, one of the two identical edges must be removed. A similar process is then applied to the root, to ensure that the root is a context

[44] With respect to some set of dimensions **D**, where $\mathbf{D} \supseteq \mathbf{D}_{min}$, and $\mathbf{D}_{min}$ contains exactly those dimensions that appear in G, and none other. In the trivial case where no dimensions appear in G (context specifiers are exclusively of the form `[]` and `[-]`), a world with respect to some arbitrary **D** can be considered for reduction to OEM; the result will be identical under every possible world and set of dimensions.

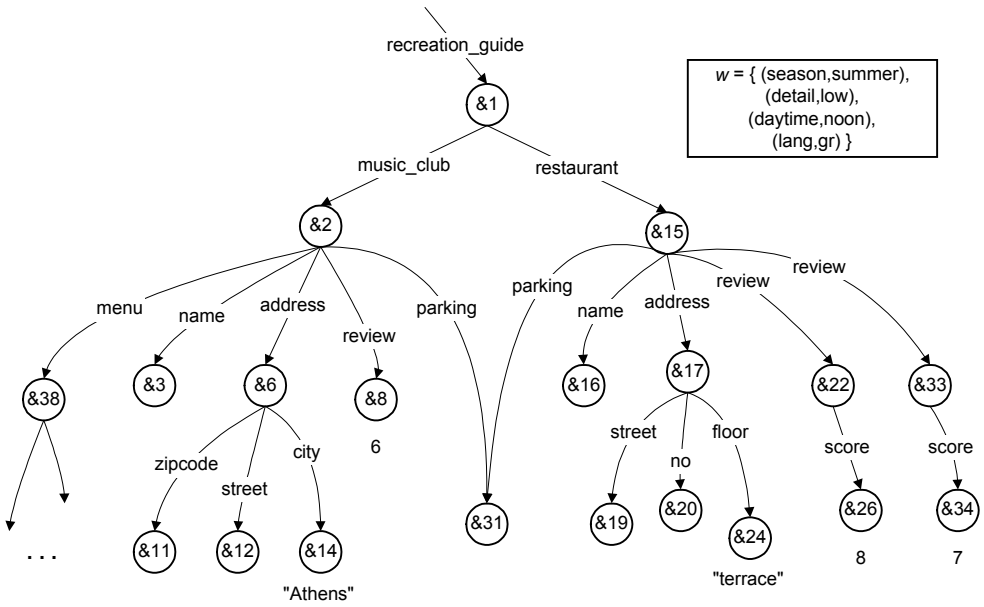
node. At the end of Step 2 every entity edge points to some context node, and every context node (except possibly the root) is pointed to by some entity edge. Finally, Step 3 gets ride of all remaining multidimensional nodes and context edges. Function v, that assigns values to nodes of the initial graph G, can be used to assign values to nodes of O_w as well.

As explained in Section 4.3.2.3, if a world w is not covered by the inherited coverage of the root r of G, it is not possible to extract an OEM holding under w from G. In this case, after Step 1 of the process there will not be any node or edge left to formulate a subgraph of G, and an empty graph will be returned.

In process **reduceToOEM** we assumed a context-deterministic Multidimensional Data Graph. For a graph G that is context-nondeterministic, it may be that under a world w more than one context nodes will correspond to a single entity edge in Step 2. In addition, we may discover that there are more than one candidate roots. In such cases, it is possible to construct more than one OEM that hold under the same world, by taking all possible combinations of candidate nodes and creating an OEM for each such combination. Another possibility is to construct a single OEM containing all holding facets of an entity as different objects. *Context-deterministic graphs ensure that for each world there exists at most one OEM holding under that world*. In the frame of this work we are mainly interested in context-deterministic Multidimensional Data Graphs.

As an example of reduction to OEM, consider the Multidimensional Data Graph in Figure 4.3. By applying the process **reduceToOEM** to that graph for the world[45] `[season=summer, detail=low, daytime=noon, lang=gr]`, we get the OEM graph depicted in Figure 4.10. All leaves are atomic nodes, however for simplicity we have omitted some of their values. Notice how the two `parking` multidimensional entities with oids &28 and &29 are represented by the object with oid &31. Also, notice how the `comment` object with oid &35 is excluded from the resulting OEM.

Figure 4.10: The OEM produced by reducing the Multidimensional Data Graph of Figure 4.3 under the world w.



[45] For simplicity, we use the term **world** to refer to a context specifier representing exactly one world.

The time complexity of process **reduceToOEM** is estimated as follows. Step 1 involves one operation of context subset for every node and every edge in the graph, therefore assuming there are *n* nodes and *e* edges, the cost of Step 1 is (*n* + *e*) ∗ <*cost of context subset*>. Step 2 scans all entity edges to find those pointing to multidimensional nodes; therefore if the number of entity edges is *ett*, we have *ett* number of comparisons. If the mean number of consecutive context edges leading to a context node is *m*, the cost cannot exceed *ett* + *ett* ∗ *m* comparisons. The same process is performed on the root, so the total cost of Step 2 is at most (*ett* + 1) + (*ett* + 1) ∗ *m* comparisons. The total cost of the process will not exceed (*n* + *e*) ∗ <*cost of context subset*> + (*ett* + 1) + (*ett* + 1) ∗ *m* comparisons.

We consider the number of edges *e* as representative of the size of the problem, and we assume that the cost of context subset is upper bounded. Nodes cannot be more than *e* + 1 (every node but the root has at least one incoming edge), and *ett*, *m* cannot be more than *e*. Therefore, the order of time complexity is $O(e^2)$.

4.4.1.3 OEMs Holding Under Two or More Worlds

In the previous section we introduced the process **reduceToOEM** that extracts from a Multidimensional Data Graph an OEM holding under a world w. The question that arises is whether the process **reduceToOEM** can be generalized for contexts representing more than one world[46]. The generalized process would take as input a Multidimensional Data Graph G and any non-empty[47] context c, and would produce an OEM O_c holding under c.

Figure 4.11: Problems in extracting an OEM that holds under two worlds.

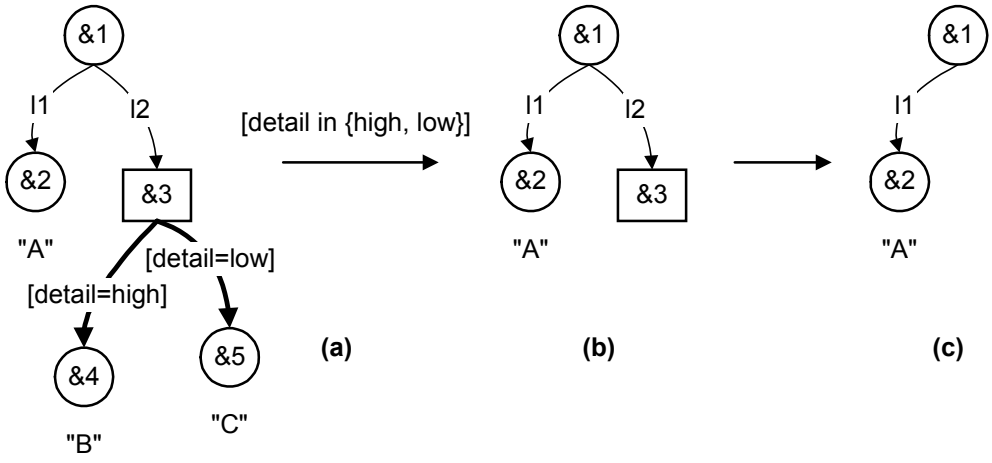


Figure 4.11 illustrates through an example how such a process would work. The graph G in Figure 4.11 (a) is to be reduced to an OEM holding under the context c = `[detail in {high,low}]`, which represents two worlds. The inherited coverages of edges in G are as follows: l1 has inherited coverage `[]`, l2 has inherited coverage `[detail in`

[46] Worlds in this section are considered with respect to $\mathbf{D}_{min}$, which contains exactly those dimensions that appear in the context specifiers of the Multidimensional Data Graph, and none other. Considering additional dimensions does not affect the problem we discuss in this section.
[47] There is no meaning in extracting an OEM that does not hold under any world.

`{high,low}]`, and the inherited coverages of the two context edges are the same as their respective explicit contexts. The subgraph of G that holds under c is shown in Figure 4.11 (b). Notice that node &3 in that subgraph is not an atomic node, but is nevertheless a leaf. Such a case could not appear if c represented a single world, because of property (c) in Section 4.3.3.2, which guarantees that leaves in the subgraph are exclusively atomic nodes of the initial graph. This property, however, does not apply to the extraction of subgraphs under more than one world.

Eliminating such illegal leaves can solve the problem, as it is shown in the OEM graph of Figure 4.11 (c). Note that this elimination may propagate upwards: suppose that edge l1 and node &2 did not exist; then eliminating l2 and &3 would leave the complex object &1 without any children, and reduction to OEM would have given an empty graph.

In addition to property (c) of Section 4.3.3.2, property (a) of the same section does not apply to the extraction of subgraphs under more than one world. The consequence is that the graph elements that hold under some context may not formulate a Multidimensional Data (sub-) Graph, if the context represents more than one world. For example, suppose that nodes &4 and &5 in Figure 4.11 (a) pointed to a new node &6. Then, node &6 would hold under the context `[detail in {high,low}]`, but no path from the root to node &6 would hold under that context.

The examples above show that process **reduceToOEM** as it is does not work for contexts representing more than one world. A generalized process would need to take into account the possibility for illegal leaves and for nodes that are inaccessible from the root. An OEM O_c holding under c would consist of graph elements that are common to all OEMs O_w, for every world w covered by c. However, O_c may not comprise every such common graph element, since some may introduce illegal leaves, while others may not be accessible from the root through a path that holds under c.

The conclusion is that our initial approach to extract O_c in a way similar to extracting O_w led us to an OEM that does not exhibit interesting properties and is not especially meaningful. The problem emanates from our assumption that O_c is formulated using those graph elements that hold under c, as is the case with single world contexts. Taking a step back, we present below a more meaningful notion of when an OEM holds under a non-empty context:

Given a Multidimensional Data Graph G and a context c, with $c \neq \mathcal{E}$*, we say that G can be reduced to an OEM* O_c *that holds under c iff G is reduced to OEM* O_c *under every world covered by c.*

Based on the above, the process for reducing G to an OEM holding under a non-empty context c comprises the following steps: (a) apply process **reduceToOEM** to G, to get the OEM O_w that holds under some world w covered by c, and (b) checking whether reduction under every other world in c would result to the same OEM O_w. Reduction to OEM under a context is obviously more restrictive than reduction to OEM under a single world. According to this latest definition, the reduction to OEM under the context `[detail in {high,low}]` of the graph in Figure 4.11 (a) is not possible[48].

A type of reduction that uses a context instead of a single world as input is **partial reduction**, which is the topic of the next section.

[48] In this case, the reduction result is not the empty graph because this would mean that reducing to OEM the initial graph under each of the two worlds in `[detail in {high,low}]` gives the empty graph as result, which is not true.

4.4.2 Partial Reduction

Given a Multidimensional Data Graph $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ and a context c, **partial reduction** extracts a subgraph of G, consisting of nodes and edges that hold under at least one of the worlds represented by c. Therefore, in contrast to reduction to OEM, the result of partial reduction is again a Multidimensional Data Graph. The following process performs partial reduction.

$G_c \leftarrow$ reducePartially (G, c) is:

1. Remove every node from V and every edge from E with inherited coverage icv, such that $icv \cap^c c = \mathcal{E}$.
2. Return $G_c = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$.

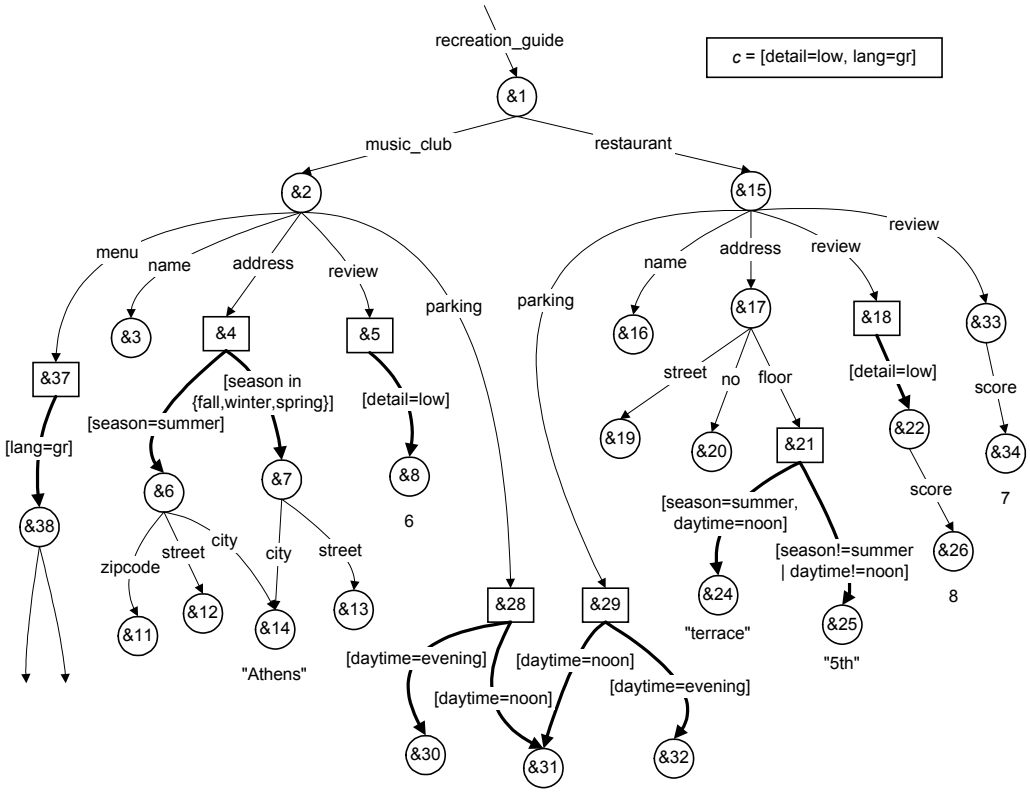
In effect, Step 1 of the process **reducePartially** removes nodes and edges of G that do not hold under any of the worlds covered by c. Remaining nodes and edges hold under at least one of the worlds in c. The graph G_c returned in Step 2 is a subgraph of G, which is also a Multidimensional Data Graph and has the same root as G. In addition, leaves in G_c are exclusively atomic nodes. To understand why, consider the following: every node or edge in G_c holds under at least some world w, therefore it is part of the subgraph G_w of G that holds under w. As shown in Section 4.3.3.2, G_w is itself a Multidimensional Data Graph with the same root as G and with atomic nodes as leaves. Obviously, all nodes and edges of G_w belong to G_c as well. Consequently, G_c is actually a conglomeration of subgraphs like G_w for every w represented by c. It follows that G_c is a Multidimensional Data Graph with the same root as G, whose leaves are atomic nodes.

It is evident that if an OEM that holds under w can be extracted from G, it can also be extracted from G_c provided that $w \subseteq^c c$. Therefore, G_c "contains" the OEMs of G that hold under every world covered by c.

Note that if c is an empty context, then the process **reducePartially** will return an empty graph. If c is a universal context, then the parts of G that will be pruned are those that do not hold under any world.

The graph in Figure 4.12 is the result of applying the process **reducePartially** to the Multidimensional Data Graph of Figure 4.3, for the context `[detail=low, lang=gr]`. Again, all leaves in the graph of Figure 4.12 are atomic nodes, but for simplicity we have omitted some of their values. Notice that the OEM graph depicted in Figure 4.10, which is the result of reduction to OEM of the graph in Figure 4.3, can also be obtained by reducing to OEM the graph in Figure 4.12. This is so because `[season=summer, detail=low, daytime=noon, lang=gr] <= [detail=low, lang=gr]`.

Figure 4.12: The subgraph produced by partial reduction of the Multidimensional Data Graph in Figure 4.3 for a context c.



The time complexity of process **reducePartially** is estimated as follows. Step 1 involves one operation of context intersection and one check for context equality for every node and every edge in the graph. Assuming there are n nodes and e edges, the total cost of the process is $(n + e) * (<\textit{cost of context intersection}> + <\textit{cost of context equality}>)$.

We consider the number of edges e as representative of the size of the problem, and we assume that the costs of context intersection and context equality are upper bounded. Nodes cannot be more than $e + 1$ (every node but the root has at least one incoming edge). Therefore, the order of time complexity is $O(e)$.

4.4.3 Reduction Primitives

It is interesting to notice that if context c represents a single world[49], Step 1 of the process **reducePartially** in Section 4.4.2 yields the same result as Step 1 of the process **reduceToOEM** in Section 4.4.1.2.

This leads us to consider a number of primitives for both reduction processes, which are listed below:

- **Graph context projection** is discussed in detail in Section 4.3.3.3.

[49] Considered with respect to $\mathbf{D}_{min}$. Additional dimensions do not play any significant role in this situation.

- **Graph context pruning** eliminates every node and edge with inherited coverage icv = $\mathcal{E}$. May return an empty Multidimensional Data Graph as a result.
- **Graph de-contextualization** is applied to a Multidimensional Data Graph G, and returns a graph without multidimensional nodes or context edges. Every entity edge in G is redirected to point to a context node, if it is not already pointing to one, as described in Step 2 of the process **reduceToOEM** in Section 4.4.1.2; when all entity edges point to context nodes, multidimensional nodes and context edges are removed from the graph.

It is obvious that the process **reducePartially** introduced in Section 4.4.2 is equivalent to graph context projection, and graph context pruning in that sequence. The only difference is that graph context projection introduces an extra multidimensional node as the new root, and a corresponding context edge labeled with the reduction context.

The process **reduceToOEM** described in Section 4.4.1.2 is equivalent to graph context projection to a single world, graph context pruning, and graph de-contextualization[50] in that sequence. Therefore, reduction to OEM is the same as de-contextualizing the result of partial reduction for a single world.

4.5 CANONICAL FORM

In Section 4.2.2 we have seen that multidimensional entities can be represented in a number of equivalent ways, and a number of examples are depicted in Figure 4.2. It follows that if two Multidimensional Data Graphs contain different but equivalent representations of multidimensional entities, then they will be different but will essentially express the same information. This observation leads to the definition of the **canonical form** (**CF** for short) of a Multidimensional Data Graph.

Definition 4.13

*A Multidimensional Data Graph is in **canonical form** iff the root is a multidimensional node, every entity edge points to a multidimensional node, and every context edge points to a context node.*

If a graph is in canonical form, every context node is the facet of some multidimensional node, and facets of multidimensional nodes are exclusively context nodes (conventional information entities).

Given a Multidimensional Data Graph G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v) we can transform G to canonical form G_{CF} through the following process.

G_{CF} ← CF (G) is:

1. If the root is a context node q, add a new multidimensional node p and a context edge *(p, [ ], q)*, and make p the new root.

[50] For de-contextualization to work correctly each entity edge must correspond to exactly one context node. As we have shown, this correspondence is guaranteed by (a) an initial graph that is context-deterministic, together with (b) graph context projection to a single world and graph context pruning. The result of de-contextualization may still not be an OEM graph: if the input graph has leaves that are not atomic nodes, the de-contextualized graph will retain those leaves. Graph context projection and graph context pruning ensure that there are not any such leaves in the input graph.

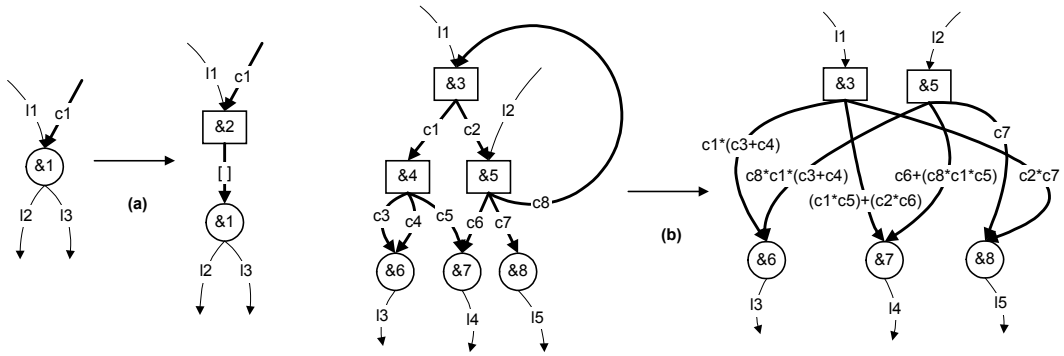
2. For every context node q pointed to by entity edge(s) of the form *(u, l, q)*, add a new multidimensional node p and a context edge *(p, [ ], q)*, and substitute every incoming entity edge *(u, l, q)* with an entity edge *(u, l, p)*.
3. For every multidimensional node q that is either the root or is pointed to by some entity edge, do the following: for every context node p accessible from q through non-cyclic paths h_1, h_2, ..., h_n of consecutive context edges, remove any context edges *(q, c, p)* if they exist, and add a new context edge *(q, c′, p)* where c′ is the context union of the path explicit contexts of h_1, h_2, ..., h_n.
4. Remove all context edges pointing to multidimensional nodes. Then, remove all multidimensional nodes that are not the root and are not pointed to by some entity edge, and all context edges departing from those nodes.
5. Return the new graph $G_{CF} = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$.

An intuition for Step 1 and Step 2 of process **CF** is given in Figure 4.13 (a), while for Step 3 and Step 4 in Figure 4.13 (b). The process does not add or delete context nodes and entity edges, although some entity edges may be redirected as depicted in Figure 4.13 (a) for l1. Multidimensional nodes and context edges may be added or deleted, but in a way that does not change the conditions that determine when a context node holds.

For example, consider entity edge l1 and node &3 in Figure 4.13 (b), which can access context node &8 through the following paths of consecutive context edges: `<(&3, c2, &5).(&5, c7, &8)>` and `<(&3, c2, &5).(&5, c8, &3).(&3, c2, &5).(&5, c7, &8)>`. In the corresponding canonical form, entity edge l1 and node &3 access context node &8 through a single context edge, whose explicit context combines the constraints expressed in the two paths: (c2 ∗ c7) + (c2 ∗ c8 ∗ c2 ∗ c7). However, note that since the second path contains a cycle and because of the additional constraints expressed by c8, the second path can only represent a subset of the worlds of the first path. Therefore, cyclic paths are excluded in Step 3, and the explicit context of the corresponding edge in canonical form becomes c2 ∗ c7.

Another point to observe is that node &4, which is not pointed by an entity edge, does not exist in the canonical form, and the conditions expressed by its respective explicit contexts have been incorporated in the explicit contexts of nodes &3 and &5.

Figure 4.13: Transforming a Multidimensional Data Graph to canonical form.



From Figure 4.13 (a) we see that Step 1 and Step 2 of the transformation do not affect the inherited context and context coverage of entity edges l1, l2, l3 and of node &1. The same is true for Step 3 and Step 4, as we can see in Figure 4.13 (b) for entity edges l1, l2, l3, l4, l5 and for nodes &6, &7, and &8. Remember that Proposition 4.3 expresses inherited context as the context union of path explicit contexts, therefore the contribution (in worlds) of edge l1 to the inherited context of node &7, for instance, remains the same after the transformation to canonical form. Similar is the case with Proposition 4.7, about the contribution of edge l4, for instance, to the context coverage of node &3. Consequently, *the inherited context and context coverage of context nodes and entity edges of the original graph is preserved in the canonical form*.

It is easy to see in Figure 4.13 that if a multidimensional node in the original graph has access under a world to more than one context nodes through consecutive context edges (Definition 4.12), the same happens in the canonical form of that graph, and vice-versa. Hence, context-determinism is maintained in the canonical form:

Proposition 4.12

If a graph is context-deterministic so is its canonical form, while if the graph is context-nondeterministic then the canonical form is also context-nondeterministic.

Proof: Obvious, from the process **CF** of transformation to canonical form.

Actually, when a graph is in canonical form it is simple to determine whether the graph is context-deterministic:

Proposition 4.13

A graph in canonical form is context-deterministic iff for every multidimensional node q the inherited coverages of edges departing from q are mutually exclusive.

Proof: Obvious.

Evidently, if the explicit contexts of those edges are mutually exclusive, the graph is context-deterministic, since the inherited coverage of an edge is context subset of its explicit context. The opposite is not true: explicit contexts may not be mutually exclusive, but the graph can still be context-deterministic. Notice that it is allowed for a multidimensional node of a context-deterministic graph to have two or more context edges with inherited coverages $\mathcal{E}$, since empty contexts are mutually exclusive with any other context, even with empty contexts.

As a conclusion, the graph G_{CF} returned by process **CF** is a Multidimensional Data Graph with the same context nodes as the original graph G, the same number of entity edges connecting corresponding (although not identical in form) entities, and equivalent constraints expressing the conditions under which facets hold. Therefore, *to every Multidimensional Data Graph G there corresponds a Multidimensional Data Graph G_{CF} that is in canonical form, and effectively contains the same information as G*.

An important consequence that stems from the above discussion is stated in the following proposition:

Proposition 4.14

*Let G be a Multidimensional Data Graph, and G_{CF} be its canonical form. Then, for every world w, if $O_w \leftarrow$ **reduceToOEM(**G, w**)** and $O_w' \leftarrow$ **reduceToOEM(**G_{CF}, w**)**, then O_w and O_w' are identical OEM graphs.*

Proof: As explained before, the canonical form maintains the inherited coverages of context nodes and entity edges, and also maintains context-determinism (Proposition 4.12). Based on that, the proposition is obvious by following the process **reduceToOEM** on G and G_{CF}.

The canonical form G_{CF} of a Multidimensional Data Graph G is free of redundant dependencies between multidimensional entities, and avoids a number of *anomalies* that may occur in G. The following refer to the graph on the left side of Figure 4.13 (b) as G:

(a) Suppose that the explicit context of the context edge *(&5, c7, &8)* of G is updated to c7′. This will not only affect the multidimensional entity represented by node &5, but the entity represented by &3 as well. This side effect is an *update anomaly*.

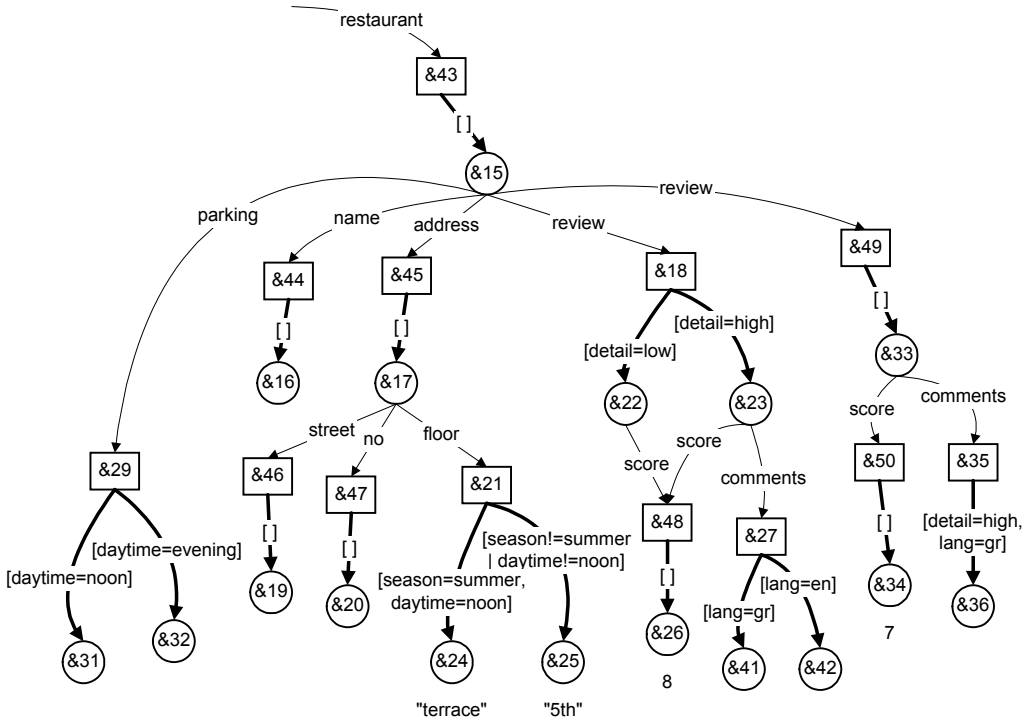
(b) If a new facet is added to the multidimensional entity represented by node &3 of G, because of edge *(&5, c8, &3)* this facet will also be indirectly added to the multidimensional entity represented by node &5. This side effect is an *insertion anomaly*.

(c) If the edge *(&3, c1, &4)* is removed from G, then the entity represented by node &4 will no longer be facet of &3. This however will cause node &5 to lose accessibility to node &6. This side effect is a *deletion anomaly*.

Those anomalies are avoided by using the canonical form G_{CF} of G, which for our examples is the graph on right side of Figure 4.13 (b).

Another property of canonical form is that *every possible path is formed by a repeated succession of one context edge and one entity edge*. There cannot be more than one consecutive context edges or consecutive entity edges in a graph that is in canonical form. This property is used for formulating and evaluating **context path expressions** in queries, which are the topic of the following chapter.

Figure 4.14: Part of the graph depicted in Figure 4.3 in canonical form.



As an example consider Figure 4.14, that shows the canonical form of the `restaurant` object in the graph of Figure 4.3.

4.6 MULTIDIMENSIONAL OBJECT EXCHANGE MODEL

Having investigated the properties of Multidimensional Data Graphs, we can now define a special type of Multidimensional Data Graph called **Multidimensional Object Exchange Model**, or **Multidimensional OEM**, or **MOEM** for short.

Definition 4.14

*A **Multidimensional Object Exchange Model** (MOEM) is a context-deterministic Multidimensional Data Graph whose every node and edge has inherited coverage icv* $\neq \mathcal{E}$.

As an example of an MOEM consider the Multidimensional Data Graph depicted in Figure 4.3. On the other hand, the graph of Figure 4.5 (a) is not an MOEM, since the inherited context and thus the inherited coverage of edge *(&5, [scale=high], &11)* and of node &11 is the empty context `[−]`.

Every node and edge of an MOEM holds under at least one world. This applies to the root as well, so the inherited coverage of the root contains at least one world; therefore an MOEM comprises at least one conventional OEM holding under some world. In addition, leaves of an MOEM must hold under a world, thus leaves are exclusively atomic nodes. Each context node and entity edge of MOEM participates in at least one conventional OEM that the

MOEM can be reduced to. Obviously, OEM is a special case of Multidimensional OEM, where there are no multidimensional nodes and context edges.

Checking that a Multidimensional Data Graph G is an MOEM involves the following steps: (a) calculate the inherited coverages of nodes and edges in G, (b) make sure that none of them is an empty context, and (c) examine G for context-determinism.

It is easy to see that the canonical form of an MOEM is again an MOEM[51]. As already pointed out, the transformation to canonical form preserves context-determinism, as well as the inherited coverages of context nodes and entity edges. Concerning the inherited coverages of multidimensional nodes and context edges, they cannot become the empty context as a result of the transformation. Moreover, as already stated, the canonical form M_{CF} of an MOEM M contains essentially the same information as M, and can be reduced to exactly the same OEMs as M.

Given a context-deterministic Multidimensional Data Graph G, we can obtain the maximum subgraph of G that is an MOEM by partially reducing G for the universal context `[]`. Partial reduction will prune nodes and edges that do not hold under any world, leaving those with inherited coverage `icv != [–]`, if any such nodes and edges exist in G. As we have seen in Section 4.3.3.2, all such surviving nodes and edges will be connected to the root through surviving paths.

4.7 MSSD-EXPRESSIONS

As pointed out in Chapter 2, the syntactic basis for expressing semistructured data is ssd-expression. We define **mssd-expression** by extending ssd-expression to incorporate context specifiers. The grammar for mssd-expression is given in Table 4.1 in Extended Backus-Naur Form [EBNF], or EBNF for short. Symbols that can be defined by a regular expression start with a capital letter (example: `AtomicValue`), while symbols defined in EBNF start with a lowercase letter (example: `complexValue`).

Table 4.1: Syntax of mssd-expression.

```
mssd-expr ::= value | Oid value | Oid
value ::= AtomicValue | "{" complexValue "}"
                      | "<" multidimValue ">"
complexValue ::= LabelName ":" mssd-expr ("," complexValue)?
multidimValue ::= cxtSpec ":" mssd-expr ("," multidimValue)?
```

It is evident that `multidimValue` corresponds to multidimensional nodes, while `AtomicValue` and `complexValue` correspond to context nodes. Conventions [ABS00] concerning the syntax of labels, and object identifiers in ssd-expressions, also apply to mssd-expressions. The grammar for context specifiers, or `cxtSpec`, is given in the previous chapter.

The mssd-expression in Example 4.1 describes the `music_club` with oid &2 in Figure 4.3.

[51] The opposite is not necessarily true: a Multidimensional Data Graph that is not an MOEM, may have a canonical form that is an MOEM. For instance, consider a Multidimensional Data Graph G containing a multidimensional node p that is a leaf and is not pointed to by any entity edge. Then G is not an MOEM, because p has empty inherited coverage. However, the canonical form G_{CF} of G does not contain p, which is eliminated by the transformation process, and G_{CF} may be an MOEM.

Example 4.1

```
&2 {menu: &37 <[lang=gr]: &38 {...},
               [lang=en]: &39 {...},
               [lang=fr]: &40 {...}>,
    name: &3,
    address: &4 <[season=summer]: &6 {zipcode: &11, street: &12,
                                      city: &14 "Athens"},
                 [season in {fall,winter,spring}]:
                                    &7 {city: &14, street: &13}>,
    review: &5 <[detail=low]: &8 6,
                [detail=high]: &9 {score: &8, comments: &10}>,
    parking: &28 <[daytime=evening]: &30,
                  [daytime=noon]: &31>
   }
```

♦

To correctly express an MOEM, the corresponding mssd-expression must be **consistent**. Consistency of mssd-expressions is similar to consistency [ABS00] of ssd-expressions. For an mssd-expression *s* to be consistent, the following must hold:

(a) Any object identifier *o* is defined at most once in *s*.

(b) If an object identifier *o* is used in *s*, it must be defined in *s*.

An object identifier is considered *defined* if it is assigned a `value`. In effect, consistency of an mssd-expression requires that every leaf node be assigned exactly one value.

While MOEM corresponds always to a consistent mssd-expression, Multidimensional Data Graph does not. In the general case, a Multidimensional Data Graph may have as leaves complex or multidimensional objects, which do not have atomic values. Using an inconsistent mssd-expression to represent such a graph may lead to ambiguity: an object in an mssd-expression that consists only of an oid without outgoing edges and without atomic value may correspond to either a leaf context node or a leaf multidimensional node in the Multidimensional Data Graph! The ambiguity is lifted if we know that the ambiguous mssd-expression represents the canonical form of a Multidimensional Data Graph, because the node type can be determined from the type of incoming edges. Moreover, the type of a node can be encoded in its oid, making the problem rather an implementation issue. In any case, a Multidimensional Data Graph whose leaves are atomic objects can be represented by mssd-expressions without any need for ambiguity resolving measures.

4.8 SUMMARY

In this chapter we incorporated context specifiers in a graph data model for semistructured data, and defined **Multidimensional Data Graph**. In Multidimensional Data Graph, conventional labeled edges (**entity edges**) define relationships between multidimensional entities, which are information entities that may manifest different facets under different worlds. Context specifiers are used to qualify those facets, and define the worlds under which a facet holds.

Contexts that are attached on the graph model refer to a specific multidimensional entity, which facilitates developing and maintenance. The actual contexts can be calculated automatically by a process that traverses the graph and takes all multidimensional entities into account. Investigating the properties of Multidimensional Data Graph, we discussed how contexts propagate through the graph, and defined:

- **Explicit context**, which is the context that qualifies facets. Explicit context expresses the worlds under which a facet *potentially* holds, and has meaning only within the boundaries of a single multidimensional entity.
- **Inherited context**, which is based on the requirement that an object may hold only under the worlds its parent object holds. Inherited context expresses the worlds under which graph elements hold, as imposed by explicit context constraints accumulated from the root to the leaves.
- **Context coverage**, which is based on the requirement that an object may hold only under the worlds it has access to some atomic node. Context coverage expresses the worlds under which graph elements hold, as imposed by explicit context constraints accumulated from the leaves to the root.
- **Inherited coverage**, which for every element of the graph represents the worlds common to inherited context and context coverage of that element. Inherited coverage states the worlds under which a graph element actually holds.

Based on context propagation, we showed that conventional OEMs holding under some world can be extracted from a Multidimensional Data Graph, and we gave the conditions under which such extractions can take place. As a part of those conditions we discussed **context-deterministic** graphs, and we presented a process that **reduces to OEM** a context-deterministic Multidimensional Data Graph under a given world. In addition, we described **partial reduction** of a Multidimensional Data Graph for a given context. Reductions are operations that may take place at any moment, while inherited coverages must be calculated only when the graph has been changed.

Redundant dependencies between multidimensional entities may cause anomalies when maintaining a Multidimensional Data Graph. **Canonical form** avoids those anomalies by transforming the initial graph into a better-structured graph.

Finally, we defined **Multidimensional OEM** (**MOEM**) as a special case of Multidimensional Data Graph that is a strict conglomeration of conventional OEMs holding under various worlds. We also specified **mssd-expression**, a syntax for expressing textually Multidimensional Data Graphs and MOEMs.

5 MULTIDIMENSIONAL QUERY LANGUAGE

In the previous chapter we proposed **Multidimensional OEM**, or **MOEM** for short, a data model for context-dependent, multifaceted semistructured data. MOEM is a special case of a graph-based data model we called **Multidimensional Data Graph**. We examined the properties of Multidimensional Data Graph in detail, and showed how MOEM relates to the OEM graph model for conventional[52] semistructured data.

In this chapter we introduce **Multidimensional Query Language**, or **MQL** for short, a query language for multidimensional semistructured data. MQL treats context as first-class citizen, and is therefore well suited for querying MOEM graphs. Although MQL can be used for querying Multidimensional Data Graphs in general[53], we are more interested in MOEMs[54]. Similarly to an OEM database [AQM+97, Suc98], we define an **MOEM database** as a database whose model is an MOEM graph. In what follows we assume that MQL queries are posed on MOEM databases, unless explicitly stated otherwise.

In the frame of the present work we do not address the issue of database updates, but focus on retrievals instead. Our goal is to demonstrate how conventional query languages can be extended to incorporate and use context in a central role. The principles used for shaping the retrieval part of MQL can also be used for designing an update part.

In what follows, we start with motivating our approach. We then discuss **context path expressions**, which are a key point of MQL. We continue with an introduction to MQL itself, followed by a discussion on more advanced issues. Finally we conclude the chapter with outlining our prototype implementation of MQL. A complete syntax specification of context path expressions and MQL is given in Appendix A.

5.1 WHY USE MOEM AND MQL?

In this section we motivate our proposal of MOEM and MQL. Simply put, a **context-driven query** is a query where context is important for selecting the right data. We present plain examples in natural language to give a first impression of context-driven queries, and of their difference from conventional queries. We argue that *when it comes to context-driven queries MQL is more powerful and more convenient to use than conventional query languages, while at the same time it becomes as simple as they are when context is not an issue*. An important point is that MQL uses the additional graph elements incorporated in MOEM in order to reply to context-driven queries.

---

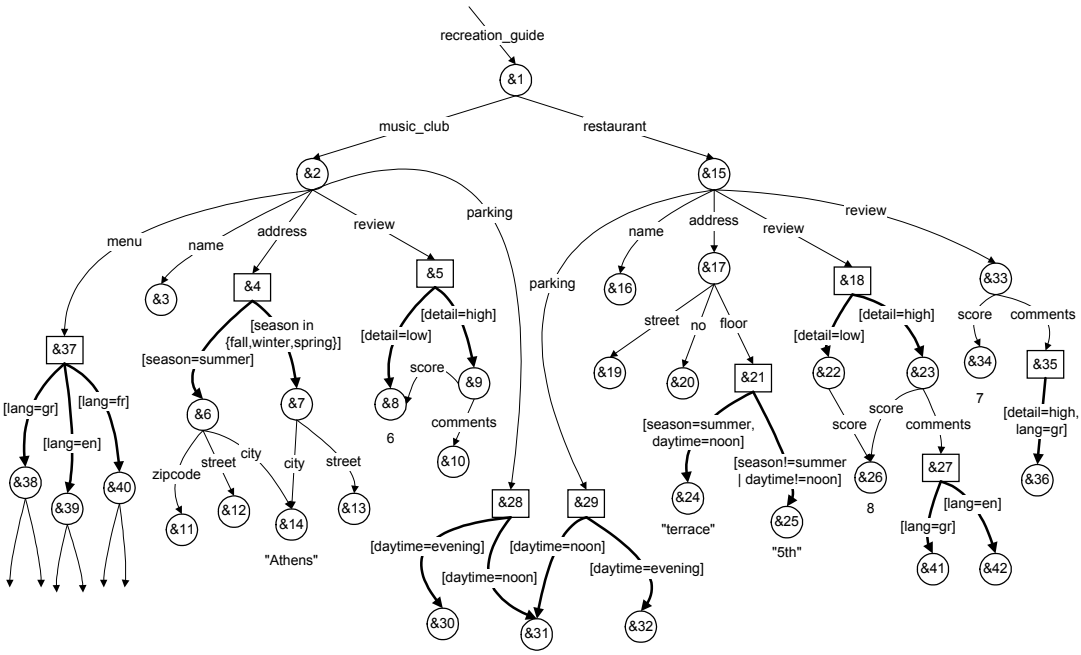
[52] By "conventional" we mean not context-dependent (or, more accurately, context-unaware).

[53] MQL does not require from the underlying graph to be context-deterministic, or to have exclusively non-empty inherited coverages. Notice that if the underlying graph has multidimensional or complex nodes as leaves (allowed in a Multidimensional Data Graph), then the result of an MQL query may also be a Multidimensional Data (sub-) Graph with multidimensional or complex nodes as leaves.

[54] For a discussion about the qualitative difference of querying MOEM and Multidimensional Data Graph, see Section 5.2.4.

For our examples we will use the context-dependent recreation guide illustrated in the previous chapter, which is depicted again in Figure 5.1. Suppose that an MOEM database M is reduced to an OEM database O_w under the world w. Then, a simple MQL query $q = (q_w, w)$ on M can be expressed as a query q_w on O_w. As an example, let M be the MOEM in Figure 5.1, and consider the query q on M "*give me the addresses of restaurants at summer noons in low detail in Greek*". Then q is equivalent to the query q_w "*give me the addresses of restaurants*" posed on the OEM facet O_w of M that holds under the world w = {(season,summer), (detail,low), (daytime,noon), (lang,gr)}. However, reducing M to O_w is not a necessary step for evaluating q; the processing of q can take place directly on M, which may be preferable in terms of performance. Intuitively, q_w can be used for navigating through MOEM entity edges, while w can guide the navigation through MOEM context edges.

Figure 5.1: An MOEM database of a context-dependent recreation guide.



In addition to such queries, the fact that Multidimensional Data Graph groups facets of entities together allows a different, "**cross-world**" type of context-driven queries. A query q of that type cannot be expressed as (q_w, w) because the information needed to answer q is not contained within a single OEM. As an example, consider the `music_club` object in M, and the following query: "*give me the name and the address in winter of a club whose summer address is provided*". Or even, another variation: "*give me the clubs that have the same address throughout the year*". This last query checks whether an address holds under a context covering all possible values of the dimension `season`. Consequently, MQL takes advantage of multidimensional nodes to pose queries that relate facets of a single multidimensional entity.

MQL is based on and extends ideas from query languages proposed for conventional semistructured data, especially Lorel [AQM+97] and UnQL [BFS00], whose underlying data model is OEM. Lorel in particular has an additional XML variant [GMW99], and is both powerful [BC00] and intuitive, features that are very appealing. As in Multidimensional OEM,

context has a primary role in MQL: path expressions combined with context information give **context path expressions**, while query conditions and construction of results may involve contexts as well. As a result, context-driven queries that would be very difficult or impossible to formulate in a query language for semistructured data, can be expressed in a compact and elegant way using MQL[55].

Naturally, it is possible to encode context-dependent information using a conventional graph model, such as OEM[56], and pose queries that contain context information using a language like Lorel. The advantage of using MQL is twofold.

(a) *Syntactic*: context-driven queries in MQL are typically much shorter and readable than equivalent Lorel queries, therefore easier to formulate and less error-prone.

(b) *Semantic*: MQL and MOEM recognize context as such and allow complex conditions that use context operations to be expressed, something that is not supported by Lorel and OEM.

The enhanced querying capabilities of MQL are partly due to the extensions to OEM that we incorporated in Multidimensional Data Graph, namely context edges and multidimensional nodes. In our view, the queries that Multidimensional OEM can answer through MQL justify the OEM extensions we introduced in the previous chapter. Moreover, *the capacity for "cross-world" queries shows that an MOEM graph is something more than the collection of OEMs it can be decomposed to*.

5.2 CONTEXT PATH EXPRESSIONS

Query languages for semistructured data use path expressions to reach to arbitrary depths in the data graph [ABS00]. Similarly, MQL incorporates **context path expressions** that define navigation patterns in Multidimensional Data Graphs. Context path expressions are the cornerstone of MQL.

Lorel distinguishes [AQM+97] between simple path expressions that allow the retrieval of objects by specifying a sequence of labels in the graph, and general path expressions, a more powerful form with advanced features such as wildcards and regular expressions. For reasons of presentation clarity, we follow an analogous distinction between **simple context path expressions** explained in this section, and **general context path expressions** described in Section 5.4.1.

This section starts with presenting context path expressions and explaining how they evaluate to nodes. We then introduce **context qualifiers**, which are used in context path expressions to state conditions over context. In what follows next, we describe how certain parts of context path expressions can be implied, leading to forms that are more readable and closer to conventional path expressions. We close this section with a discussion on the semantics of context path expressions.

[55] For a comparison between MQL and Lorel queries refer to Section 5.5.2, where we discuss translation of MQL queries to equivalent Lorel queries.
[56] An example of such an encoding can be seen in Section 5.5.1, where we use OEM to represent Multidimensional OEM.

5.2.1 Incorporating Context in Path Expressions

A simple path expression in Lorel is a sequence $X.l_1.l_2....l_n$, where $l_1, l_2, ..., l_n$ are labels, and X is an object identifier or a variable denoting an object. Similarly, **context path expressions** start with an object identifier, or a variable denoting an object[57]. In MOEM object identifiers are partitioned in two disjoint sets of context nodes and multidimensional nodes[58], which leads to two distinct types for object variables: variable X denotes a context object type, while variable <X> denotes a multidimensional object type.

Context path expressions use two kinds of components to navigate through entity edges and context edges, the **entity part** component and the **facet part** component. Entity parts start with a dot (".") and define navigation patterns over entity edges, while facet parts start with a double colon ("::") and define navigation patterns over context edges. Remember that entity edges depart only from context nodes, and context edges depart only from multidimensional nodes; therefore, dots can be seen as corresponding to context nodes, while double colons as corresponding to multidimensional nodes.

5.2.1.1 Context Path Expressions and Canonical Form

Paths in a Multidimensional Data Graph consist of a number of entity edges and context edges *in any order*. So, if entity parts and facet parts had to have a strict correspondence to "matching" edges, one would have to know the exact structure of the graph so as to form a context path expression with entity parts and facet parts in the proper order. That would be a problem, since we would like to be able to create context path expressions without precise knowledge of the succession of entity edges and context edges in a graph. In addition, as explained in the previous chapter, different graph structures can be equivalent as far as the represented information is concerned. For example, consecutive context edges in a path can be substituted by a single context edge with a suitable explicit context[59]. We would like a context path expression to match all equivalent path structures, without describing exhaustively all possible edge successions.

To achieve those goals *we build context path expressions around the canonical form of a Multidimensional Data Graph*. We quote from the previous chapter that, if a Multidimensional Data Graph is in canonical form, "...every possible path is formed by a repeated succession of one context edge and one entity edge. There cannot be more than one consecutive context edges or consecutive entity edges in a graph that is in canonical form". Building context path expressions around the canonical form means that we have a fixed order of entity and context edges, against which to match entity and facet parts. Furthermore, it means that a context path expression has *identical evaluation* for all graphs that have the same canonical form.

Consequently, simple context path expressions consist of an object identifier or object variable, followed by a number of entity parts and facet parts succeeding one another. A *context* object identifier or variable should be immediately followed by an entity part, while a *multidimensional* object identifier or variable should be followed by a facet part.

---

[57] In Section 5.2.1.1 we explain that a context path expression may also start with an **entity part** corresponding to the root of the graph.
[58] As stated in the previous chapter, the terms **node** and **object** are used interchangeably.
[59] As explained in the previous chapter, this explicit context is given by the context intersection of the explicit contexts of the consecutive context edges.

Example 5.1

The following p1 to p5 are possible forms of simple context path expressions, where .e_i indicates entity parts and ::f_i indicates facet parts.

p1→ $o_{cxt}.e_1::f_1.e_2::f_2....e_n::f_n$
p2→ $X.e_1::f_1.e_2::f_2....e_n$
p3→ $o_{mld}::f_1.e_2::f_2.e_3::f_3....e_n::f_n$
p4→ $<X>::f_1.e_2::f_2.e_3::f_3....e_{n-1}::f_{n-1}.e_n$
p5→ $e_{root}::f_1.e_2::f_2.e_3::f_3....e_n::f_n$
♦

As shown in the context path expressions of Example 5.1, entity parts and facet parts alternate successively. Context path expression p1 starts with the context object identifier o_{cxt}, followed directly by the entity part .e_1 that is matched against entity edges departing from o_{cxt}. Context path expression p2 starts with the context object variable X instead of an identifier. Notice that we numbered entity and facet parts as pairs (pair 1, pair 2, etc.), starting with the entity part. The intuition is that such a pair "corresponds" to a single label .l_i of conventional path expressions. In an analogous way to p1, example p3 starts with a multidimensional object identifier o_{mld}, followed by the facet part ::f_1 that is matched against context edges departing from o_{mld}, while p4 starts with the multidimensional object variable <X> instead of an identifier. A context path expression ends with either an entity part or a facet part.

The case of p5 demonstrates a handy convention in both OEM and MOEM: the root of the graph is represented by a unique edge that points to the root but does not depart from any node. That edge is labeled with the name of the database, as in Figure 5.1 where the root is pointed to by an edge labeled recreation_guide. The root of the canonical form of a Multidimensional Data Graph is always a multidimensional node, thus the edge in question is always an entity edge, as implied by the entity part e_{root} in p5. Note that in context path expressions e_{root} is a special entity part without the ".", while in conventional path expressions the "label" of the root is considered an object identifier.

5.2.1.2 Evaluation of Context Path Expressions

A context path expression evaluates either to a set of context objects or to a set of multidimensional objects. This depends on the last part of the context path expression: if the last part is a facet part, as in p1, p3, and p5 of Example 5.1, then the context path expression evaluates to a set of context nodes; on the other hand, if the last part is an entity part, as in p2 and p4, then it evaluates to a set of multidimensional nodes.

In OEM, a data path [AQM+97] is a sequence $o_0,l_1,o_1,l_2,o_2,...,l_n,o_n$, such that there exists an edge l_k from object o_{k-1} to object o_k. Given an OEM graph, there may be none, one, or several such data paths that match the simple path expression $X.l_1.l_2....l_n$ for $X = o_0$. Similarly to data paths, a **context data path** in Multidimensional Data Graph is a sequence $o_0,[icv_1]l_1,o_1,[icv_2]l_2,o_2,...,[icv_n]l_n,o_n$, such that there exists an edge with label l_k and with inherited coverage $[icv_k]$ from object o_{k-1} to object o_k. Note that if the edge is a context edge, then the label is its explicit context. Although inherited coverages are not necessary for identifying edges, they are useful for matching the context data path against a context path expression, as explained later. Given a Multidimensional Data Graph, there may be none, one, or several context data paths that match a context path expression. The last nodes of the matching context data paths form the result set of the context path expression.

Although we assumed that context path expressions target MOEM databases that are in canonical form, it is important to realize that it is not necessary to transform a graph to

canonical form in order to evaluate a context path expression. When a graph is in canonical form, context data paths exhibit the same regularity as context path expressions, with an entity edge succeeding a context edge repeatedly. In this case there is a direct correspondence between the individual components of context data paths and context path expressions, which facilitates the matching process. In the general case however where a graph is not in canonical form, the matching process can adjust the correspondence between the individual components of context data paths and context path expressions to "simulate" a canonical form[60]. As we will see in Section 5.2.3, certain parts of context path expressions can be omitted, which complicates things further. To keep things simple, in what follows we assume that MOEM databases are in canonical form.

5.2.1.3 Entity Parts and Facet Parts

As pointed out, to determine whether a context data path matches a context path expression, entity edges are compared against entity parts and context edges are compared against facet parts in sequence. An **entity part** is either of the form `.l` or of the form `.[cq`$_{icv}$`]l`, where `l` is an entity edge label and `[cq`$_{icv}$`]` is an **inherited coverage qualifier**. A **facet part** is either of the form `::[cq`$_{ec}$`]` or of the form `::[cq`$_{icv}$`][cq`$_{ec}$`]`, where `[cq`$_{ec}$`]` is an **explicit context qualifier** and `[cq`$_{icv}$`]` is again an inherited coverage qualifier. Explicit context qualifiers constitute a necessary component of facets parts, and are matched against the explicit context of corresponding context edges. Inherited coverage qualifiers are optional for both entity parts and facet parts, and are matched against the inherited coverage of a path, or *path inherited coverage* (defined in the previous chapter). Inherited coverage qualifiers and explicit context qualifiers have the same syntax, and are instances of **context qualifiers**. Context qualifiers are constructs used to express conditions on contexts. In the basic case, a context qualifier is just a context specifier denoting the worlds that a matching context must cover.

Example 5.2

The following context path expressions refer to (the canonical form of) the graph in Figure 5.1.

```
p1→ [lang=gr]recreation_guide::[-].music_club::[-].menu::[-]
p2→ [-]recreation_guide::[-].music_club::[-].menu::[lang=gr]
p3→ [-]recreation_guide::[-].music_club::[-].menu::[-]
p4→ []recreation_guide::[-].music_club::[-].address
p5→ []recreation_guide::[-].music_club::[-]
                                        .review::[-][detail=high]
```

♦

In the context path expression `p1` of Example 5.2, the context qualifier `[lang=gr]` is compared to the path inherited coverage of the paths that start from the root and conclude at a `menu` facet. Specifically, it matches those path inherited coverages that are *context superset* of `[lang=gr]`. The other context qualifiers in `p1` are explicit context qualifiers, and match any context edge, because any context is context superset of the empty context `[-]`. On the MOEM database of Figure 5.1, `p1` evaluates to the set {&38}. In the next case, `p2` also happens to evaluate to {&38}; the inherited coverage qualifier is now `[-]`, which matches any path inherited coverage, while the last explicit context qualifier is `[lang=gr]`, which

[60] For example, a facet part may be matched against a *sequence* of consecutive context edges of a graph that is not in canonical form. A subtle point is that the canonical form may contain additional multidimensional nodes, which can be the target of context path expressions. An evaluation process that assumes a graph is *not* in canonical form must implement a consistent oid scheme for such "implied" multidimensional nodes.

matches the explicit contexts that are *context superset* of `[lang=gr]`. Although `p1` and `p2` happen to evaluate to the same result for the given MOEM database, they are conceptually different: `p1` returns the `menu` facets that are accessible from the root through a path that holds (at least) under `[lang=gr]`[61], while `p2` returns the `menu` facets with corresponding explicit context (at least) `[lang=gr]` irrespective of whether those facets actually hold under some world(s). In the case of `p3` all context qualifiers are empty contexts, therefore every menu facet matches the context path expression, and the result is {&38, &39, &40}. Note that no grouping of the results takes place: if the music club in Figure 5.1 contained more than one `menu` multidimensional objects, the matching `menu` facets would be returned in a flat result set, with no indication of which objects in the set are facets of the same menu.

The context path expression `p4` in Example 5.2 returns a set of *multidimensional* nodes, specifically the set {&4}. The inherited coverage qualifier `[]` at the beginning of `p4` demands that a path leading from the root to an `address` multidimensional node must hold under every possible world. The case of `p5` shows that a context path expression can contain many inherited coverage qualifiers. It looks for `review` objects that are accessible through a path holding under every possible world, and then returns the `review` facets that have corresponding explicit context (at least) `[detail=high]` irrespective of the worlds under which they actually hold. The first context qualifier `[]` of `p5` is an inherited coverage qualifier, and is compared against the path inherited coverage of paths that conclude at `review` multidimensional nodes. The second inherited coverage qualifier is `[-]`, and qualifies context edges that depart from `review` multidimensional nodes. In this case the path consists of a single context edge. The meaning of this second inherited coverage qualifier in `p5` is that context edges pointing to `review` facets may actually hold under any set of worlds. Therefore, *an inherited coverage qualifier marks the start of the path it qualifies and the end of the path for the previous inherited coverage qualifier* in the same context path expression.

Context qualifiers make context path expressions somewhat verbose and difficult to follow, but as we will see in Section 5.2.3 in most cases they can be implied, leading to easier and more compact forms of context path expressions. Before explaining the cases where context qualifiers can be omitted, we present context qualifiers in detail.

5.2.2 Context Qualifiers

Context qualifiers are used in context path expressions to state conditions that contexts associated with edges or with paths must satisfy. A context qualifier can take the form of a context specifier, a **context pattern**, or a **context variable**.

5.2.2.1 Context Qualifiers as Context Specifiers

A context qualifier that has the form of a context specifier c_{cq} expresses a condition against some context c, stating that for c_{cq} and c to match, c must be *context superset* of c_{cq}. In other words, all worlds represented by the context qualifier must be covered by a matching context[62]. According to this, the context path expression `o.[`c_1`]lb::[`c_2`][`c_3`]` matches the

[61] In other words, `p1` returns `menu` nodes that exist in each conventional OEM holding under the worlds specified by `[lang=gr]`. This is explained in more detail in Section 5.2.4.

[62] This is more restrictive than requiring that the two contexts should contain at least one world in common. Let alone the trivial case of empty contexts, in the canonical form of a *context-deterministic* graph, the condition that the inherited coverage of a context edge must be context superset of some context can be met by at most one context edge within any multidimensional entity. So, in the case of

paths that, starting from o, consist of an entity edge labeled lb with inherited coverage[63] icv_{ett} >= [c_1], followed by a context edge with inherited coverage icv_{cxt} >= [c_2] and explicit context ec_{cxt} >= [c_3].

Since the empty context specifier [-] is subset of any context, when used as a context qualifier it plays the role of a **context wildcard** that matches any edge context. Using the empty context in such a way essentially "turns off" context conditions. For example, in the context path expression X.[-]lb::[c][-] the entity part .[-]lb will match all entity edges labeled lb irrespective of their inherited coverage, while the facet part ::[c][-] will match all context edges with an inherited coverage that is superset of [c] irrespective of their explicit context. As another example consider <X>::[-][-].lb, where a context edge is allowed to have any explicit context, and together with an lb entity edge they may have any path inherited coverage. According to the above, the facet part ::[-][-] will match any context edge.

As already stated, inherited coverage qualifiers qualify paths that consist of one or more edges, and therefore they may affect subsequent entity and facet parts. For example, in the context path expression X.[c_1]lb::[c_2] the context qualifier [c_1] is not matched against the inherited coverage of lb, but against the path inherited coverage of a path made of: (a) an entity edge labeled lb, and (b) a context edge with explicit context covering [c_2]. It is however interesting to observe that, if an inherited coverage qualifier has the form of a context specifier, it can be repeated throughout its corresponding path without changing the evaluation of a context path expression. Based on this, if [c_1] and [c_3] are context specifiers, the context path expression X.[c_1]a::[c_2].b::[c_3][c_4].d is equivalent to the context path expression X.[c_1]a::[c_1][c_2].[c_1]b::[c_3][c_4].[c_3]d, in which every inherited coverage qualifier is matched against the inherited coverage of a single edge. This is shown by the following proposition:

Proposition 5.1

Let p be a path in a Multidimensional Data Graph, and c be a context qualifier that has the form of a context specifier. Then, c matches the path inherited coverage of p iff c matches the inherited coverages of all edges in p.

Proof: If c matches the inherited coverages of all edges that comprise p, then c is context subset of each of those inherited coverages. This means that c is context subset of the context intersection of those inherited coverages. Therefore, c matches the path inherited coverage of p. Conversely, if c matches the path inherited coverage of p, then c is context subset of the path inherited coverage of p. This means that c is context subset of the inherited coverage of every edge in p. Therefore, c matches the inherited coverages of all edges in p.

It must be emphasized that this property of inherited coverage qualifiers only holds if they have the form of *context specifiers*. Inherited coverage qualifiers that are context patterns or context variables do *not* give equivalent context path expressions when repeated throughout the path they qualify.

---

context-deterministic graphs (and assuming non-empty contexts) at most one context edge can match an inherited coverage qualifier. As a result, this "more restrictive" approach allows context qualifiers to uniquely specify facets of multidimensional entities.

[63] When a path consists of a single edge, its path inherited coverage is the inherited coverage of the edge.

5.2.2.2 Context Patterns

Context patterns allow to express contexts by specifying only some of the dimensions, while the rest can take any value. The syntax of context patterns is identical to that of context specifiers, with the difference that: (a) the character "%" follows the opening bracket "[" signifying a context pattern, and (b) a clause in the context pattern must contain at least one dimension specifier, and cannot be "–" (empty clause) or "" (universal clause)[64]. The specified pattern is not applied at a syntactic level, but *at the level of worlds*, as demonstrated by the following example.

Example 5.3

Let:

$\mathbf{V}_{lang}$ = {en, gr, sp}
$\mathbf{V}_{detail}$ = {low, high}
$\mathbf{V}_{format}$ = {ps, pdf, html}

c_1 = [detail=low, lang in {en,gr}, format=pdf]
c_2 = [detail=low, format=pdf | lang=en, format=ps |
lang in {gr,sp}]

Then, the following are context conditions that use context patterns:

$cond_1$: [% lang=gr] <= c_1 *(true)*
$cond_2$: [% lang in {gr,en}, detail=low] = c_1 *(true)*
$cond_3$: [% lang in {gr,en}, detail=low] = c_2 *(false)*
$cond_4$: [% lang=en, detail=high | lang=gr, detail=low] <= c_2 *(true)*

♦

In Example 5.3, condition $cond_1$ holds true. It demands that context c_1 covers some world where language is Greek, irrespective of the value of other dimensions. The symbol <= of context subset declares that lang may take values different than gr in other worlds covered by c_1, as long as in some world of c_1 the value of lang is gr. Condition $cond_2$ also holds true. It states that c_1 must cover: (a) worlds that have language Greek and low detail, together with (b) worlds that have language English and low detail, irrespective of the value of other dimensions. The symbol = of context equality implies that *every* world in c_1 must fall in one of those two categories. Condition $cond_3$ is false, because context c_2 covers worlds that do not match the pattern, as for example the world {(lang,en), (detail,high), (format,ps)}. Condition $cond_4$ is true. It uses a context pattern with two clauses, and demands that c_2 covers some world where language is English and detail is high, *plus* another world where language is Greek and detail is low[65]. In a context condition, a context pattern may be compared against a context specifier for context equality / inequality, or (proper) context subset / superset.

A straightforward way to evaluate context conditions that involve context patterns is to transform them to equivalent context conditions that do not contain context patterns. Informally, we remove the character "%" from the context pattern, and at the same time we remove all dimension specifiers from the other part (the context specifier) that do not

[64] A context pattern can contain empty clauses and universal clauses. For instance, dimension specifiers like d in {} and d in ALL can be used for the respective cases.

[65] Notice that context patterns with many clauses, like the one in $cond_4$, do not differ from patterns having only one clause. As with context specifiers, many clauses are used when a set of worlds cannot be expressed by a single clause.

correspond to some dimension in the context pattern[66]. For instance, $cond_3$ of Example 5.3 will give `[lang in {gr,en}, detail=low] = [detail=low | lang=en | lang in {gr,sp}]`, which is a context condition solely between context specifiers. This context equality is not true, therefore the initial condition that involves the context pattern does not hold.

Context patterns can also participate in context intersection, context union, and context difference. In this case, *context patterns are used to "mask" context specifiers*, screening out dimensions and interfering with dimension values as shown below.

Example 5.4

Let:

$\mathbf{V}_{lang}$ = {en, gr, sp}
$\mathbf{V}_{detail}$ = {low, high}
$\mathbf{V}_{format}$ = {ps, pdf, html}

The following are examples of context operations that involve context patterns:

```
1. [% lang in {gr,sp}] * [lang in {en,gr}, detail=low] = [lang=gr]
2. [% lang=gr, detail=low] + [lang=en, detail=low, format=html] =
                                        [lang in {gr,en}, detail=low]
3. [format in {pdf,ps}, lang in {en,sp}] - [% lang=sp] = [lang=en]
4. [% format in {pdf,ps}] – [detail=low, format=ps] = [format=pdf]
```

♦

As with context conditions, in the context operations of Example 5.4 we remove the character "%" from the context pattern, and at the same time we remove all dimension specifiers from the context specifier that do not correspond to some dimension in the context pattern[67]. Then, we carry out the context operation as if between two context specifiers.

5.2.2.3 Context Qualifiers as Context Patterns

When a context pattern is used as a context qualifier, it asserts the condition that the corresponding edge or path context must be context superset of the context pattern.

For example, the context path expression `o.[%c1]lb::[c2][%c3]` matches the paths that, starting from `o`, consist of an entity edge labeled `lb` with inherited coverage icv_{ett} `>= [% c1]`, followed by a context edge with inherited coverage icv_{cxt} `>= [c2]` and explicit context ec_{cxt} `>= [% c3]`. As another example consider `o::[%c1][c2].lb::[%c3]`, where `[%c1]` is the only inherited coverage qualifier and is matched against the path inherited coverage of the complete path[68].

Referring to the MOEM database of Figure 5.1, the context path expression

---

[66] Care should be taken not to remove dimension specifiers of the form (*dimension*, ∅) whose presence makes the clause an empty clause, but to substitute the whole clause with the empty clause {–} instead.

[67] Again, care should be taken not to remove dimension specifiers of the form (*dimension*, ∅) whose presence makes the clause an empty clause, but to substitute the whole clause with the empty clause {–} instead.

[68] As pointed out, in contrast to context specifiers, if `[%c1]` is repeated throughout the path it qualifies (namely in front of `lb` and `[%c3]`) the resulting context path expression will not be equivalent to the original. To see why, let `[%c1]` be `[% lang=gr]` and the inherited coverages of two of the edges in the path be `[lang=gr, detail=low]` and `[lang=gr, detail=high]`. Then the path inherited coverage will be an empty context that does not match the given context pattern, although both the inherited coverages of the individual edges match the context pattern.

```
[% daytime=noon]recreation_guide::[-].restaurant::[-]
                                          .address::[-].floor::[-]
```

seeks the `floor` facets where, occasionally, restaurants operate at noons, and evaluates to {&24, &25}. Node &24 is included because the restaurant operates there at summer noons, while node &25 is included because of autumn, winter, and spring noons. Notice that if the context specifier `[daytime=noon]` had been used instead of the context pattern, the result would have been an empty set.

5.2.2.4 Context Qualifiers as Context Variables

Apart from context object variables, denoted `X`, and multidimensional object variables, denoted `<X>`, context path expressions use **context variables**[69], that bind to context specifiers and are denoted `[X]`. A context variable that appears in the place of an inherited coverage qualifier or explicit context qualifier binds to the corresponding path inherited coverage or explicit context, respectively.

For example, lets consider the context path expression `o.[X]lb::[c][Y]`, which contain the context variables `[X]` and `[Y]`. This context path expression matches the paths that, starting from object `o`, consist of an entity edge labeled `lb` followed by a context edge with inherited coverage that is context superset of `[c]`. Lets assume that the context path expression matches the following context data paths: cdp_1 leading to node n_1, and cdp_2 leading to node n_2. Then, the result of the context path expression will be the set of triplets $\{(x_1, y_1, n_1), (x_2, y_2, n_2)\}$, where x_i is the value of `[X]` and y_i is the value of `[Y]`. Specifically, x_i is the inherited coverage of the edge labeled `lb` in the context data path cdp_i, and y_i is the explicit context of the following context edge.

Another example yet is `o.[X]a::[`c_1`].b::[`c_2`][`c_3`]`, where the context variable `[X]` binds to the path inherited coverage of paths that start from `o` and include in sequence: an entity edge labeled `a`, a context edge with explicit context that covers `[`c_1`]`, and an entity edge labeled `b`. So, in a way, context variables are similar to label variables of UnQL [BFS00], with the difference that they bind to contexts instead of labels. Context variables are used in MQL to specify complex conditions on contexts, and can also participate in the construction of new graphs that are the result of queries.

5.2.3 Implied Context Qualifiers

To simplify context path expressions and make them more readable, context qualifiers can be omitted in certain cases. When a context qualifier is missing, the default value is assumed. The *default value for context qualifiers* is the empty context `[-]`, which does not impose any restrictions on edge or path contexts.

Implying context qualifiers is demonstrated through the following example.

Example 5.5

In each pair, the two context path expressions are equivalent: in the first a number of context qualifiers are implied, while in the second the implied context qualifiers appear explicitly. The context qualifiers `[`c_i`]` that appear below can be anything from context specifiers, context patterns, and context variables.

1. `X.a::[`c_1`].b::[`c_2`][`c_3`].d`
   `X.[-]a::[`c_1`].b::[`c_2`][`c_3`].d`

[69] Notice the difference between **context variable** and **context object variable**.

2. X.[c_1]a.b.d
   X.[c_1]a::[-].b::[-].d
3. <X>.[c_1]a.b::[c_2][-].d
   <X>::[-][-].[c_1]a::[-].b::[c_2][-].d

♦

In Example 5.5 (1), the first occurrence of an inherited coverage qualifier is [c_2] and appears in the facet part ::[c_2][c_3]. In this case, the inherited coverage qualifier for the unqualified part of the path (.a::[c_1].b) is implied, and is the default context qualifier [-]. Pair (2) shows a context path expression where the facet part that should exist between .[c_1]a and .b is missing altogether. The facet part that should exist between .b and .d is missing as well. In this case, the default context qualifier [-] is assumed as the explicit context qualifier of the implied facet parts. In pair (3), context path expressions start with <X>, which is a multidimensional object variable and should be followed by a facet part. This facet part is missing, implying the facet part ::[-][-], which matches any context edge. Therefore, pair (3) combines the cases shown in pairs (1) and (2). Another interesting point in pair (3) is the facet part ::[c_2][-]. Although the explicit context qualifier is [-], it cannot be omitted, because the facet part introduces the inherited coverage qualifier [c_2] at that position. If [-] was missing, then [c_2] would be taken for an explicit context qualifier.

While facet parts can be implied, *entity parts cannot be implied*. Therefore, the following are not legal context path expressions:

- X::[c_1][c_2]
- <X>::[c_1][c_2]::[c_3][c_4]

Similarly to query languages for semistructured data, in MQL a context path expression may be followed by an object variable that binds to the set of the result nodes. The fact that facet parts may be missing raises again the issue discussed in Section 5.2.1 of the type of nodes a context path expression evaluates to. As mentioned in Section 5.2.1, if the last part is a facet part then a context path expression evaluates to a set of context nodes, whereas if the last part is an entity part then it evaluates to a set of multidimensional nodes. Thus, a facet part may be present at the end of a context path expression just to signify that the result should be a set of context nodes, which is not very elegant. Fortunately, we can still omit this facet part, and use an object variable and coercion to achieve the same goal, as demonstrated in the example that follows.

Example 5.6

Each pair consists of two context path expressions: the first is followed by an object variable that imposes a coercion on the type of result nodes, while the second is an equivalent context path expression where coercion does not take place. Omitting the object variable Y in the first context path expression of a pair changes its evaluation to multidimensional nodes.

1. X.[c_1]a::[c_2].b Y
   X.[c_1]a::[c_2].b::[-] Y
2. X.a.b.d Y
   X.[-]a::[-].b::[-].d::[-] Y

♦

Example 5.6 shows that a context path expression followed by a context object variable always evaluates to context objects, irrespective of whether it ends with an entity or a facet part. More specifically, a context object variable after a context path expression that concludes to an entity part, implies the addition of the facet part ::[-] at the end of the context path expression. Therefore, *a context object variable can cause a coercion at the level of context path expression*.

Given that entity parts cannot be implied, multidimensional object variables cannot cause a coercion analogous to context object variables. Expressions where a multidimensional object variable is next to a facet part, as for example in `X.[c1]a::[c2][c3] <Y>`, are illegal expressions, since coercion of the context path expression is not possible. Nevertheless, as we have seen in Example 5.5 (3), a multidimensional object variable causes a facet part to be implied when at the beginning of a context path expression.

It is interesting to note that the context path expression of Example 5.6 (2) has the form of a conventional path expression: all context qualifiers and facet parts are missing. This implies that *conventional path expressions are a special case of context path expressions*, where context conditions have been "turned off"[70].

Example 5.7

The context path expressions below are followed by an object variable, and refer to the MOEM database in Figure 5.1.

```
p1→ [lang=gr]recreation_guide.music_club.menu  X
p2→ recreation_guide.music_club.menu::[lang=gr]  X
p3→ recreation_guide.music_club.menu  X
p4→ []recreation_guide.music_club.address  <X>
p5→ []recreation_guide.music_club.review::[-][detail=high]  X
p6→ [% daytime=noon]recreation_guide.restaurant.address.floor  X
p7→ recreation_guide.restaurant.[X]parking  Y
p8→ recreation_guide.restaurant.parking::[X][-]  Y
```

♦

In Example 5.7, cases `p1` to `p5` are equivalent to the corresponding cases `p1` to `p5` of Example 5.2, with the difference that context qualifiers that can be implied are missing. The explanation that follows Example 5.2 is still relevant here, but a comparison shows immediately that context path expressions of Example 5.7 are much more intuitive and easier to formulate. Case `p6` has been explained in Section 5.2.2.3, and `p7` evaluates to the set {(`[daytime=noon]`, &31), (`[daytime=evening]`, &32)}, where the first value of each tuple corresponds to variable `[X]` and the second value to variable `Y`. Case `p8` happens to give the same result as `p7`, but in `p7` the path where `[X]` binds to consists of a `parking` entity edge and a succeeding context edge, while in `p8` the path consists of the context edge only. Notice that, although good practice requires the use of different names for variables, the two variables `[X]` and `Y` in cases `p7` and `p8` are of different type and could in principle use the same identifier, `[X]` and `X` respectively, without introducing ambiguities.

To summarize, the following simple steps can be used to complete the missing context qualifiers in a context path expression.

1. *Explicit context qualifiers*: for each facet part that is implied (as determined by the proper succession of entity and facet parts and by coercion to context object variables), the default value `[-]` is assumed.
2. *Inherited coverage qualifiers*: if the first part of a context path expression, be it an entity part or a facet part, does not contain an inherited coverage qualifier then the default value `[-]` is assumed.

[70] In the previous chapter we mentioned that OEM is a special case of Multidimensional Data Graph. If O is a conventional OEM with M its multidimensional counterpart, and p_O is a conventional path expression with p_M its multidimensional counterpart, then it is easy to see that evaluating p_O on O gives the same results as evaluating p_M on M.

The following section aims to give an insight on context path expressions by attempting a qualitative analysis of their meaning.

5.2.4 Semantics of Context Path Expressions

In this section we interpret context path expressions in terms of conventional path expressions and OEMs, and explain the comparative potential of inherited coverage qualifiers and explicit context qualifiers when used on MOEMs and Multidimensional Data Graphs.

We start with the following context path expression *cpe*:

```
[c]a.b.d
```

Lets assume that *cpe* evaluates to a node *q* on a Multidimensional Data Graph M. We also assume that `[c]` is a context specifier that represents the worlds $\{w_1, w_2, ..., w_n\}$, and that reducing M under each of these worlds gives the OEMs $O_1, O_2, ..., O_n$ respectively. Then, node *q* is also returned by the (conventional) path expression `a.b.d` evaluated on each and every of $O_1, O_2, ..., O_n$. Here is why: because of the properties of path inherited coverage, if a node *q* is returned by *cpe*, then edges `a`, `b`, and `d` and the relevant context nodes of the corresponding *context data path* form a data path that survives in every $O_1, O_2, ..., O_n$. We can therefore say that a context path expression *cpe* of the form `[c]a.b.d` has a conventional path expression counterpart `a.b.d`, whose evaluation on every $O_1, O_2, ..., O_n$ is superset of the evaluation of *cpe* on M.

To see which of the nodes `a.b.d` evaluates to belong to the evaluation of *cpe*, we must examine the corresponding data paths in every $O_1, O_2, ..., O_n$. If a data path leading to *q* exists in every $O_1, O_2, ..., O_n$, then there exists a corresponding *context data path* in M and *q* is also returned by *cpe*. A point that requires attention is that *the same data path* must survive in $O_1, O_2, ..., O_n$. Figure 5.2 illustrates this through an example.

Figure 5.2: Data paths do matter.

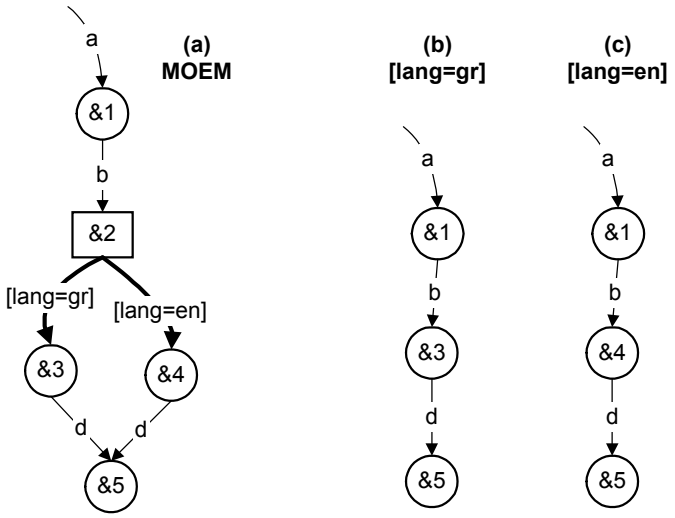


Figure 5.2 (a) depicts the Multidimensional Data Graph M, which happens to be an MOEM, while (b) and (c) depict the OEMs that hold under the worlds {(lang,gr)} and

{(lang,en)} respectively. The only difference between (b) and (c) is the oid of the middle node, which is &3 in (b) and &4 in (c). The path expression `a.b.d` returns {&5} when evaluated on either (b) or (c), but the context path expression `[lang in {gr,en}]a.b.d` returns an empty set when evaluated on (a). Consequently, `[c]a.b.d` does not evaluate to the nodes that are just common in the result sets of `a.b.d` against O_1, O_2, ..., O_n; the corresponding data paths (which include the oids of traversed nodes) must be identical as well.

Lets now consider the context path expression *cpe₁*:

$[c_1]a.b.[c_2]d$

This context path expression can be comprehended if we consider the following two context path expressions, *cpe₂* and *cpe₃* respectively:

$[c_1]a.b \quad X$
$X.[c_2]d \quad Y$

Both *cpe₂* and *cpe₃* are similar to *cpe*, and the discussion about *cpe* applies to *cpe₂* and *cpe₃* as well. The object variable `X` binds to matching nodes of *cpe₂*. Each matching node is treated as root by *cpe₃*, which returns the nodes accessible through the edge `d` under all worlds specified by $[c_2]$. Obviously *cpe₁* is equivalent to the pair of *cpe₂* and *cpe₃*, in the sense that *cpe₁* evaluates to the set of nodes `Y` binds to[71].

Context path expressions like *cpe₁* can be thought of as specifying a path not only between nodes, but also between sets of worlds, "jumping" to different OEM reductions. Actually, the context path expressions that we discussed in this section *can be viewed as joins between OEM nodes and between OEM data paths that hold under different worlds*.

A related issue is the position of a context qualifier in a context path expression. Consider a database of employees with context representing valid time, and the context path expressions *cpe₁*, *cpe₂*, and *cpe₃* respectively:

```
[validtime=30]db.company.employee.salary  X
db.company.employee.[validtime=30]salary  X
db.company.employee.salary::[validtime=30][-]  X
```

Context path expression *cpe₁* evaluates to `salary` context objects for which there exists a path `db.company.employee.salary` in the conventional OEM that holds under time instance 30. Context path expression *cpe₂* is less restrictive than *cpe₁*, and allows the path leading to employee facets to hold under any world[72]. Specifically, *cpe₂* requires that a path `db.company.employee` exist in some OEM that holds under any time instance, and that `salary` edges exist in the OEM that holds under time instance 30. The join is performed on employee facets from which `salary` edges depart. A node *q* returned by *cpe₂* exists in the

---

[71] Note that it is also possible to write *cpe₁* as

$[c_1]a \quad Z$
$Z.[c_1]b \quad X$
$X.[c_2]d \quad Y$

or to break it even further using facet parts and multidimensional object variables, *but* only if $[c_1]$ and $[c_2]$ are context specifiers: as shown in Proposition 5.1, an inherited coverage qualifier that is a context specifier can be repeated throughout the path it qualifies. However, if $[c_1]$ is a context pattern or a context variable, the corresponding path `a::[-].b::[-]` must be qualified as a whole. In a similar way, if $[c_2]$ is a context pattern or a context variable, the corresponding path `d::[-]` must be qualified as a whole.

[72] If the database is an MOEM, the path will hold under *some* world.

OEM that holds under the time instance 30, as does the `salary` edge pointing to it and the node from which that edge departs, but the path from this node to the root of the particular OEM is not specified. Context path expressions like cpe_2 focus on specific parts of an MOEM, and qualify with a context only some of the objects (and their relations) in a path.

Note that if the path inherited coverage of every instance in the database of the path `db.company.employee::[-]` is context superset of `[validtime=30]`, then cpe_1 and cpe_2 are equivalent. The explanation is that, in this case, for each node q returned by cpe_2, the data path that corresponds to `db.company.employee` and exists in some OEM will also exist in the OEM that holds under time instance 30, and q will be returned by cpe_1 as well. Because of this we may use the form of cpe_2 in queries instead of cpe_1, usually when the inherited coverage of the first edges in the database is the universal context; such edges may be left without an inherited coverage qualifier.

The above discussion applies to cpe_3 as well. The difference from cpe_2 is that cpe_3 requires that only the salary facet exist in the OEM holding under time instance 30, and that this node is accessible through a path `db.company.employee.salary` in any OEM that holds under some time instance. On the other hand, cpe_2 requires that the salary facet *and* the `salary` edge *and* the corresponding employee facet exist in the OEM holding under time instance 30.

Another interesting issue is the comparative expressiveness of non-empty context qualifiers on MOEMs and on Multidimensional Data Graphs. Inherited coverage qualifiers are strong with MOEMs, where the inherited coverages are never empty contexts. On the other hand, in Multidimensional Data Graphs there may exist subgraphs with empty inherited coverage (meaning that they do not hold under any world), which essentially reduces the navigational possibilities of context path expressions. Using non-empty inherited coverage qualifiers only, a context path expression can access solely the MOEM sub-part of a Multidimensional Data Graph.

However, as pointed out in the previous chapter, there are meaningful queries in Multidimensional Data Graphs involving facets that do not hold under *any* world (but that still have the *potential* to hold under some world, as they have non-empty explicit context). Such queries can be formulated using explicit context qualifiers, which give a "low-level" navigation capability to context path expressions.

The context path expressions that we examined in this section evaluate only to context objects. The reason is that *it is not possible to render the semantics of a multidimensional object using conventional OEMs*.

5.3 MQL BASICS

Multidimensional Query Language (**MQL** in short) is based on query languages for semistructured data, in particular Lorel [AQM+97], which it extends by using context path expressions and additional clauses that manipulate context. To concentrate on context-related issues, we used as a starting point a basic "core language" described in [ABS00] that incorporates essential features from Lorel and UnQL [BFS00], while at the same time it is simple and easy to extend. MQL retains the main characteristics of Lorel, namely powerful path expressions and type coersions. Since the main aim of MQL is the formulation of context-driven queries, we do not focus here on features that are explained in depth elsewhere, like for example coercion rules. MQL semantics are relegated to Lorel semantics (and ultimately to ODMG [ODMG]); in Section 5.5 we implement MQL on top of LORE, and specify how MQL queries can be translated to equivalent Lorel queries.

MQL comprises the following clauses:

```
select  <results template>
context <context variable definitions>
from    <context path expressions>
where   <predicate>
within  <context predicate>
```

The `select` clause must exist in an MQL query, whereas the rest of the clauses are optional. Generally speaking, the `from` clause results in variable bindings and should be considered first; the `where` and `within` clauses filter the bound values and are considered next; the `context` clause defines new context variables that are used in the `select` clause; finally, the `select` clause is considered, which constructs the result graph. In what follows, we describe the clauses one by one.

5.3.1 "from" Clause

The role of the `from` clause is to introduce variables attached to context path expressions. The output of a `from` clause is a set of tuples, each tuple containing variable bindings for a matching context data path.

```
from recreation_guide.restaurant X,
     X.[season=winter]address.floor Y,
     X.[season=summer,daytime=noon]address.floor Z
```

When evaluated on the MOEM database of Figure 5.1, this `from` clause gives a set of a single tuple {(&15, &25, &24)}, where tuple values correspond to variables (`X`, `Y`, `Z`).

The same `from` clause can be written as:

```
from recreation_guide.restaurant{X}.[season=winter]address.floor Y,
     X.[season=summer,daytime=noon]address.floor Z
```

As in Lorel, curly brackets `{}` are used to introduce object variables in the middle of long paths. Object variables in curly brackets can be context `{X}`, or multidimensional `{<X>}`. Multidimensional object variables cannot appear immediately after a facet part, as explained in Section 5.2.3.

An important point is that, if the inherited coverage qualifier `[c]` is a context specifier, because of Proposition 5.1 the `from` clause

```
from [c]l1.l2 X
```

is equivalent to

```
from [c]l1         <V1>,
     <V1>::[c][-]  V2,
     V2.[c]l2      <V3>,
     <V3>::[c][-]  X
```

However, if `[c]` is a context variable or a context pattern, the two `from` clauses above are *not* equivalent, because `[c]` refers to the *path* inherited coverage of `l1::[-].l2::[-]`, not

to individual edges[73]. Using curly brackets is a convenient way to introduce object variables in the middle of context path expressions and at the same time allows context variables and context patterns to qualify long paths as a whole.

Notice that in the following case

```
from [c]l1.l2 X,
     X.l3.l4 Y
```

the context qualifier `[c]` that qualifies the first path does not qualify the second, and the default inherited coverage qualifier is implied: `X.[-]l3.l4 Y`. In case we use object variables embedded in the context path expression

```
from [c]l1.l2{X}.[-]l3.l4 Y
```

we need to include the inherited coverage qualifier `[-]` before `l3`, otherwise `[c]` will qualify the complete path.

5.3.2 "select" Clause

The `select` clause constructs the results. The result of an MQL query is a Multidimensional Data Graph in the form of an mssd-expression, which is constructed based on a template provided in the `select` clause. When the query

```
select restaurant: {name: <Z>, winter_floor: Y}
from recreation_guide.restaurant X,
     X.[season=winter]address.floor Y,
     X.name <Z>
```

is evaluated on the database of Figure 5.1, it returns the mssd-expression:

```
{restaurant: {name: <[]: &16 "...">, winter_floor: &25 "5th"}}
```

Figure 5.3 (a) gives the results of the query pictorially. For simplicity the graph in Figure 5.1 does not display a value for the leaf node &16 that represents the name of the restaurant, so we write `"..."` instead of that value. Notice the use of the multidimensional object variable `<Z>`, which binds to an implied multidimensional node introduced by the transformation of the MOEM in Figure 5.1 to canonical form. This also explains the universal context `[]` in the results.

To facilitate comparisons, we indicate in query results the oids of objects as they appear in the database of Figure 5.1. Keep in mind however, that *oids in query results are not necessarily the same as oids of the corresponding objects in the database*.

---

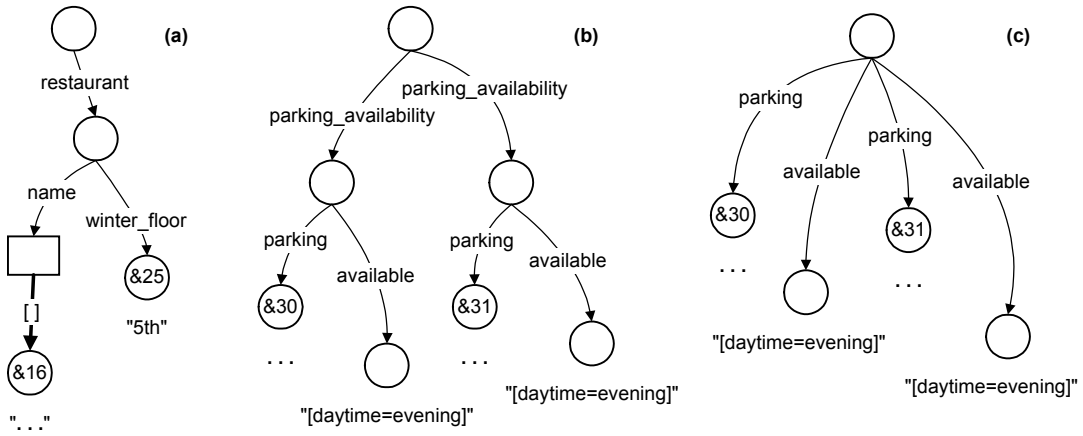
[73] If `[V]`, `[V1]`, `[V2]`, `[V3]`, `[V4]` are context variables, then the expression

```
from [V]a.b X
within predicate([V])
```

is equivalent to the expression

```
from [V1]a <X1>, <X1>::[V2][-] X2,
     X2.[V3]b <X3>, <X3>::[V4][-] X
within predicate([V1]*[V2]*[V3]*[V4])
```

Figure 5.3: Results of MQL queries as Multidimensional Data Graphs.



Variables that appear in `select` are "projected" from the variable bindings coming out of the `from` clause, resulting in a multiset (bag) of new tuples, or a set if the keyword `distinct` is used. For instance, in case two different `restaurant` objects (different values for `X`) point to the same `name` and `floor` objects, then we will have two identical (`<Z>`, `Y`) tuples in the results, unless the keyword `distinct` follows `select`. Duplicate elimination for objects is based on oid.

As we have seen, the mssd-expression template in the `select` clause can contain multidimensional object variables and context object variables in the place of object values. Moreover, it can contain context variables, as shown in the following query:

```
select parking_availability: {parking: X, available: [Y]}
from recreation_guide.music_club.parking::[Y] X
```

The context variable `[Y]` binds to the explicit contexts that correspond to `parking` facets, and the result of the query is the mssd-expression

```
{parking_availability:
          {parking: &30 {...}, available: "[daytime=evening]"},
 parking_availability:
          {parking: &31 {...}, available: "[daytime=noon]"}}
```

The results[74] are also shown as a graph in Figure 5.3 (b). Notice that values of the context variable `[Y]` become string values of objects in the result[75]. A similar query follows:

```
select parking: X, available: [Y]
from recreation_guide.music_club.parking::[Y] X
```

The result of the query is:

[74] For brevity, the graph in Figure 5.1 is not complete. Here we assume that parking facets are complex objects. If we consider them to be atomic instead, { . . . } should be replaced by " . . . ".

[75] A similar case in Lorel is when using label variables or path variables to convert metadata (label names) to data (string values of objects in the result).

```
{parking: &30 {...},
 available: "[daytime=evening]",
 parking: &31 {...},
 available: "[daytime=noon]"}
```

Figure 5.3 (c) depicts the results of the query as a graph. Observe that if `parking_availability` is omitted, then objects `parking` and `available` are no longer grouped in pairs. The reason is that variables `X` and `[Y]` are bound in pairs, but unless a parent object is used to reflect this in the results, the information about which `X` value corresponds to which `[Y]` value is lost.

Context variables can also be used in the place of context edge labels in the mssd-expression template of `select`, specifying the explicit context of context edges in the results. Section 5.4.3 discusses advanced features of `select`, including omission of labels and use of context path expressions, and examines more complex cases of constructing MQL results.

5.3.3 "where" Clause

The `where` clause does not differ from its Lorel counterpart, and filters results in exactly the same way:

```
select restaurant: {name: P, winter_floor: Y}
from recreation_guide.restaurant X,
     X.[season=winter]address.floor Y,
     X.[season=summer,daytime=noon]address.floor Z
     X.name P
where Z="terrace"
```

The above query returns the name of a restaurant and the floor it operates on in winter, for restaurants that operate on a terrace on summer noons. It is important to realize that, when a restaurant points to a `name` multidimensional object that comprises two facets, the restaurant will participate as two different `restaurant` objects in the result of the query, one for each `name` facet. This happens because MQL does not group objects automatically, but creates objects in the results according to the tuples of variable bindings. Grouping of objects is discussed in Section 5.4.2.

Another interesting point is the use of coercion in the comparison `Z="terrace"`, exactly like in Lorel. Variable `Z` is bound to an object, but is compared to a string. Normally this comparison would always give `false` because of type mismatch, but because of coercion it returns `true` if the value of the object currently bound to `Z` is equal to the string[76]. Moreover, an address that does not contain a floor facet holding under summer noons will not cause a type error, but will simply fail the condition[77]. Coercion facilitates the formulation of queries on data with varying structure (irregular types and missing fields). Coercion is used in MQL in the same way as in Lorel: (a) it alleviates the need to know whether a value is a number or a string and allows to compare a value with an object reference, (b) it allows to compare an

[76] When = is used between two objects, coercion does not take place and equality is determined by comparing oids. As in Lorel, the operator == can be used to force coercion and comparison of values instead of comparison of oids.

[77] As in Lorel, the special value *nil* indicates that there is no object binding for a variable, a common case in semistructured data where structure is varying. The presence of *nil* in a condition always makes the condition false. In addition, a *nil* value does not cause the creation of a new object in the result graph.

object with a set of objects, implying an existential operator. The following query shows an example of this:

```
select name: Y
from recreation_guide.restaurant{X}.name Y
where X.review.score::[-] >= 8
```

The expression `X.review.score::[-]` ends with a facet part and evaluates to a set of context objects, which is compared against the numeric value `8`. The facet part `::[-]` is not necessary, because MQL forces evaluation to context objects whenever context path expressions are compared against strings or numeric values. The query is interpreted as:

```
select name: Y
from recreation_guide.restaurant{X}.name Y
where exists Z in X.review.score (Z >= 8)
```

A facet part `::[-]` at the end of the context path expression in `where` is again optional, because the context object variable `Z` forces evaluation to context objects. The above query does not use contexts; the implied context qualifiers default to `[–]`, and they do not impose any conditions on contexts. Note that a Lorel query with the same meaning would be syntactically very similar to this particular MQL query: *when the MQL context features are not used, MQL queries become as simple to formulate as Lorel queries*.

In effect, context path expressions in `where` imply (parts of) a `from` clause. As an example consider:

```
select club_with_menu: {X.name, <Y>}
where exists recreation_guide.-{X}.menu{<Y>}
```

The query uses a context path expression in `select`, while some labels are omitted from the mssd-expression template. Both those issues are addressed in Section 5.4.3.2. The wildcard – matches any entity edge label, and is discussed in Section 5.4.1.2. Variables introduced as part of context path expressions in `where` can be used in other clauses as well, like for example `X` and `<Y>` in `select`. The context path expression in `where` implies a `from` clause:

```
select club_with_menu: {X.name, <Y>}
from recreation_guide.-{X}.menu{<Y>}
where exists <Y>
```

When evaluated on the database of Figure 5.1, the `form` clause binds (`X`, `<Y>`) to the tuples {(&2, &37), (&15, *nil*)}. For the binding of `X` to &15 that corresponds to `restaurant` of Figure 5.1, variable `<Y>` binds to the special value *nil*, because the `restaurant` object does not contain a `menu`. Thus, the second tuple fails the condition in `where`, and `select` returns one `club_with_menu` object. If the condition in `where` had not been present, a second `club_with_menu` object containing just the `name` of object &15 would have been included in the results.

5.3.4 "within" Clause

The `within` clause plays the role of a `where` clause for contexts: it expresses context conditions on context variables that are introduced in other clauses. The context predicate must be satisfied by a tuple of variable bindings, otherwise the tuple is filtered out.

```
select row: {context: [X], menu: Y}
from recreation_guide.music_club.menu::[X] Y
within [X] * [lang in {gr,en}] != [-]
```

The query uses the context variable `[X]` to transform context metadata in the database to normal data in the result graph, but also to filter out unwanted nodes in the `within` clause. It returns the menus of music clubs in Greek or English, together with their corresponding explicit context.

Similar to the `where` clause, the context predicate in the `within` clause may contain context variables, context specifiers, and context patterns, together with the context operation symbols `+`, `*`, `–`, `<`, `>`, `<=`, `>=`, `=`, `!=`, brackets, and the keywords `and`, `or`, and `not`. A complete specification of the syntax of `within` clause is given in Appendix A.

5.3.5 "context" Clause

The `context` clause defines new context variables that can be used subsequently in the `select` clause for constructing the result graph. The values of the newly introduced context variables are calculated separately for every tuple of variable bindings that has passed the `select` and `within` clauses, and the results are inserted in the corresponding tuple and augment it.

```
select menu_worlds: [Z]
context [Z] as union([Y])
from recreation_guide.music_club{X}.menu::[Y][-]
where oid(X) = "&2"
```

The above query uses the **context aggregate function** `union([c])`, to compute the worlds under which there exists a `menu` facet for the music club with oid &2. The function `oid(OBJVAR)` returns the oid of an object as a string. When evaluated on the MOEM of Figure 5.1, the result is the mssd-expression:

```
{menu_worlds: "[lang in {gr,en,fr}]"}
```

As another example, consider the query

```
select comments: {language: [Z], contents: X}
context [Z] as [% lang in ALL] * [Y]
from recreation_guide.#.comments::[Y][-] X
```

which returns for each `comments` facet, the actual comments and their language. The symbol `.#` is a wildcard that matches any path, and is explained in Section 5.4.1.2. The keyword `ALL` is shorthand for the complete domain of a dimension. The query binds the context variable `[Y]` to the inherited coverages of context edges leading to `comments` facets. The inherited coverage may contain dimensions other than `lang`, like for example the dimension `detail`. Those dimensions are screened out in the `context` clause, using context intersection with the context pattern `[% lang in ALL]`. In this way, `[Z]` contains only the languages under which a facet holds, leaving out other dimensions. Context patterns are used in context intersection, context union, and context difference to "mask" context specifiers, as described in Section 5.2.2.2.

The `context` clause may define any number of context variables separated by commas. A context variable defined in the `context` clause cannot appear at the right side of the keyword `as` of any variable definition in the same clause. In a variable definition, the right side can

contain context variables, context specifiers, context patterns, the context operation symbols +, *, –, brackets, and the context aggregate functions `union([c])`, and `intersect([c])`[78].

Moreover, the function `extension([c])` can be used, which turns a context into "worlds". In reality, the context specifiers returned by `extension([c])` may not represent worlds in the strict sense, because the function assumes that the set of dimensions **D** contains only the dimensions that appear in its current context argument.

Specifically, the function `extension([c])` produces a new set of context specifiers for every context bound to the context variable `[c]` as follows: (a) if the parameter value is *nil* or if it contains the universal clause ∅, a set containing only *nil* is returned, (b) if the parameter value is an empty context, an empty set is returned, and (c) in any other case, a set of context specifiers that correspond to the *context extension* of the parameter value is returned. The following example shows `extension` used with a context as argument:

```
extension([lang in {gr,en},detail=low]) = {[lang=gr,detail=low],
                                           [lang=en,detail=low]}
```

Lets assume `[c]` is a context variable, and let F_i be the result set of `extension([c])` for the value v_i of `[c]`, $1 \le i \le n$. Let S_i be the subset of tuples of variable bindings before `extension([c])` was called, for which `[c]` has the value v_i. Then, the new set of tuples of variable bindings after calling `extension([c])` is $\sum_{i=1}^{n} (F_i \times S_i)$. The symbol $\times$ represents the Cartesian product, and $\sum_{i=1}^{n}$ the set union of all Cartesian products from i = 1 to i = n.

In Section 5.4.3.3 we demonstrate the use of `extension([c])` through an example query. A complete specification of the syntax of `context` clause is given in Appendix A.

5.3.6 Dimensions and Dimension Domains in MQL

As shown in previous chapters, it is not necessary to know the actual set of dimensions **D** in order to carry out the context operations used in MQL. Therefore, MQL does not have to know the set of dimensions **D** for a database, but computes context operations based on the context specifiers at hand.

However, we have seen that for certain context operations it is necessary to know the dimension domains for the dimensions involved. We assume that, given a dimension, MQL has complete information (or access to information) concerning the domain of that dimension.

5.4 ADVANCED ISSUES

Some features of query languages for semistructured data are orthogonal to context-driven queries and can be easily incorporated in MQL, while others require more attention. As our intention is to show how context can be incorporated in a query language, we focus on the latter. In this section we continue the presentation of MQL with a few key issues: general context path expressions, nested MQL queries, and construction of results.

---

[78] The parameter `[c]` can be a bound context variable or a context specifier. In case `[c]` is a context specifier, `union([c])` and `intersect([c])` return `[c]`.

5.4.1 General Context Path Expressions

General context path expressions extend **simple context path expressions** with a more powerful syntax that gives MQL the flexibility to pose queries without exact knowledge of the database structure. General context path expressions feature **regular expressions** and **wildcards**.

In this section we also introduce MQL **label variables** and **path variables**, which although not part of general context path expressions are closely related to them.

5.4.1.1 Regular Expressions

Like the "core language" presented in [ABS00], MQL uses **regular expressions** [Fri97] at two levels: at the level of entity edge labels, and at the level of entity parts and facet parts[79]. The example that follows looks for `review` facets of objects whose name contains "music" or "Music":

```
X.[% detail=high]".*[Mm]usic.*".review::[-]
```

The use of quotes signifies that regular expressions apply at the level of labels rather than at the level of entity parts and facet parts. Therefore, `".*[Mm]usic.*"` matches labels such as: `music_club`, `MusicRestaurant`, `live-music-hall`, etc. (`.*` means any sequence of zero or more characters). The use of the context pattern `[% detail=high]` denotes that the corresponding path inherited coverage must contain at least a world where detail is high.

The next example uses regular expressions at the level of entity parts and facet parts, and looks for `menu` facets in Greek of music clubs and restaurants:

```
[% lang=gr]recreation_guide(.music_club | .restaurant).menu::[-]
```

Regular expressions at the two levels can be combined, as shown in the example below:

```
X.theater(."zip.*" | .[% season=summer]address."(code)|(zip.*)")
```

In this context path expression, if the label of an entity edge departing from some theater facet[80] starts with "zip", then the inherited coverage qualifier is implied and defaults to `[–]` for the whole path. Alternatively, a theater facet may lead to an `address` edge and the corresponding address facet be followed by a `code` edge, or by an edge with a label that starts with "zip". In this case, the inherited coverage qualifier of the path that starts with `address` is `[% season=summer]`.

Using regular expressions at the level of entity edge labels is straightforward; on the other hand, the level of entity parts and facet parts requires more attention. If s_1 and s_2 are legal sequences of entity parts and facet parts, a **general context path expression component** (for short **gcpe_component**) may have the forms:

$s_1 s_2$ $\quad$ $s_1 \mid s_2$ $\quad$ (s_1) $\quad$ $(s_1)?$ $\quad$ $(s_1)+$ $\quad$ $(s_1)*$

The symbol | means a disjunction, ? means 0 or 1 occurrences, + means 1 or more occurrences, and * means 0 or more occurrences.

---

[79] A third level is that of contexts, where we would like to express constraints on certain dimensions without fully defining a context specifier. This is covered by **context patterns**, presented in Section 5.2.2.2.

[80] Remember that, although this context path expression does not contain any facet parts, facet parts are implied. The `theater` entity edge is followed by some context edge leading to a theater facet.

To avoid the formulation of illegal context path expressions, like in `X.label(::[c])+` where facet parts are stacked one after another, we distinguish four kinds of gcpe_components depending on the type of their initial and final parts:

- `e-e`, which starts with an entity part and ends with an entity part.
- `e-f`, which starts with an entity part and ends with a facet part.
- `f-f`, which starts with a facet part and ends with a facet part.
- `f-e`, which starts with a facet part and ends with an entity part.

Then, $s_1 s_2$ is legal only if the initial part of s_2 is allowed to follow the final part of s_1, while the disjunction | can only be used between gcpe_components of the same kind. The symbols ?, +, and ∗ can be used with gcpe_components `e-f` and `f-e`, while for `e-e` and `f-f` only `(e-e)+` and `(f-f)?` is allowed.

Generally speaking, variables cannot be introduced in gcpe_components. The only type of variable that can appear in a gcpe_component is context variables placed as inherited coverage qualifiers (context variables cannot appear as explicit context qualifiers in gcpe_components). In case a context variable appears in a disjunction, as in `(.[X]a | .[Y]b)`, and depending on the matching context data path, its value may be the special value *nil*[81].

Concerning the symbols + and ∗, inherited coverage qualifiers in a gcpe_component *are used only in the first occurrence* and are not repeated in subsequent occurrences[82]. For instance, the gcpe_component `(.[% quality=high]part)+` signifies expressions of the form `.[% quality=high]part.part....part`. Notice that the inherited coverage qualifier `[% quality=high]` qualifies the complete path and is not repeated for each individual `part` entity edge.

The Kleene closure ∗ introduces a problem when used on graph databases that contain cycles: a regular expression terminating to ∗ or + may match an infinite number of data paths. There are a number of ways to deal with this issue, one of which is not to allow crossing the same object twice when matching such a regular expression [AQM+97].

5.4.1.2 Wildcards

Wildcards are useful when some of the labels in a path are not known, or their relative position in a path is not known. We have already mentioned that the empty context qualifier `[–]` behaves as a **context wildcard**, matching any context. Analogously, the wildcard – matches any entity edge label. As an example consider the following context path expression, which uses the wildcard – in a regular expression, and matches any low detail `review` facet in the database, however deeply nested:

---

[81] Similarly to *nil* for object variables, the presence of *nil* in a context condition always makes the context condition false. In addition, a *nil* value for a context variable does not cause the creation of a new object in the result graph. Context operations with *nil* evaluate to *nil*, while the presence of *nil* in context aggregate functions is ignored. As already mentioned, `extension(`*nil*`)` returns a set containing only *nil*.

[82] As we have shown in Section 5.2.2.1, repeating an inherited coverage qualifier throughout the path it qualifies results in an equivalent context path expression *only if the qualifier has the form of a context specifier*. If the qualifier is a context pattern or a context variable, then the new context path expression is not equivalent to the original. By using inherited coverage qualifiers only in the first occurrence, we allow context patterns and context variables in regular expressions to qualify paths instead of individual edges.

```
recreation_guide(.–::[-])*.review::[% detail=low][-]
```

The wildcard # is used to denote a path consisting of zero or more entity edge / context edge pairs. In particular, `.#` stands for `(.-::[-])*`, and is an `e-f` gcpe_component signifying expressions of the form `.-::[-].-::[-]....-::[-]`. On the other hand, `::#` is shorthand for `(::[-].-)*`, and is an `f-e` gcpe_component that signifies expressions of the form `::[-].-::[-].-...::[-].-`[83]. Using #, the example above can be written as:

```
recreation_guide.#.review::[% detail=low][-]
```

The wildcard # can be preceded by an inherited coverage qualifier: the `e-f` gcpe_component `.[c]#` stands for `(.[c]-::[-])*`, and signifies expressions of the form `.[c]-::[-].-::[-]....-::[-]`, while the `f-e` gcpe_component `::[c]#` stands for `(::[c][-].-)*`, and signifies expressions of the form `::[c][-].-::[-].-...::[-]:-` that have 0 or more facet part / entity part pairs.

5.4.1.3 Path Variables and Label Variables

MQL incorporates path variables as encountered in Lorel, and label variables as encountered in UnQL.

Path variables are of the form `@X` and bind to **context data paths** (defined in Section 5.2.1.2). They can be used in the `where` clause to test context data paths for equality, or in the `select` clause to output context data paths in the query results. A bound path variable `@X` can also appear in the function `pathof(@X)`[84], which strips a context data path of oids and inherited coverages, and transforms it to a sequence of entity edge labels and context edge labels of the form "`...a::[ec1].b::[ec2]...`". If a path variable appears in `select`, a new node is created in the result graph for each bound path, having the path as its string value.

```
select distinct with_menu_in_greek: pathof(@P)
from recreation_guide(.#)@P.menu::[% lang=gr]
```

The above query returns the paths that lead to `menu` multidimensional objects that comprise a Greek facet. The part of a context path expression that corresponds to a path variable must be enclosed in brackets, as in `(.#)@P`. The path variable `@P` binds to matching context data paths, which start from a root facet and end to a context node from which a `menu` edge departs. Since many different context data paths for different clubs can give the same path (sequence of edge labels), we use `distinct` with `pathof(@P)` for duplicate elimination. On the MOEM of Figure 5.1 only one path is returned, which consists of one entity edge and one context edge.

```
{with_menu_in_greek: "music_club::[]"}
```

The following query uses wildcards and path variables to detect edges in the database that do not hold under *any* world.

---

[83] Since the implied final part of `::#` is an entity part, the expression `::#::f` is legal (where `::f` is a facet part). Similarly, because the implied final part of `.#` is a facet part, the expression `.#::f` is illegal.

[84] For a path variable `@X`, the analogous function in Lorel is `pathof(X)`, not repeating the character `@`. In contrast to Lorel, MQL considers `@` as part of the path variable identifier.

```
select icv_violation: @P
from root::#(::[X][-])@P
within [X]=[-]
union
select icv_violation: @P
from root.#(.[X]-)@P
within [X]=[-]
```

The query is the union of two separate queries. The first query uses a path variable to bind to any *context edge* with empty inherited coverage, while the second uses a path variable to bind to any *entity edge* with empty inherited coverage. The keyword `union` merges the roots of the two result graphs into a common root, and then eliminates duplicate edges[85] departing from the new root. The form of the final results is depicted in Figure 5.4 (a). The value of the first object corresponds to a context edge, while the value of the second to an entity edge; `[icv]` is the empty inherited coverage, `[ec]` denotes some explicit context, and `label` denotes some entity edge label.

In a Multidimensional Data Graph, if a node has empty inherited coverage then incoming and outgoing edges have empty inherited coverage as well. Therefore, if the query above does not return any `icv_violation` object, the database does not contain any edge or node with empty inherited coverage. To ensure that the database graph is actually an MOEM, we also need to check whether it is context-deterministic. The following query returns pairs of context edges that violate context-determinism, if such pairs exist:

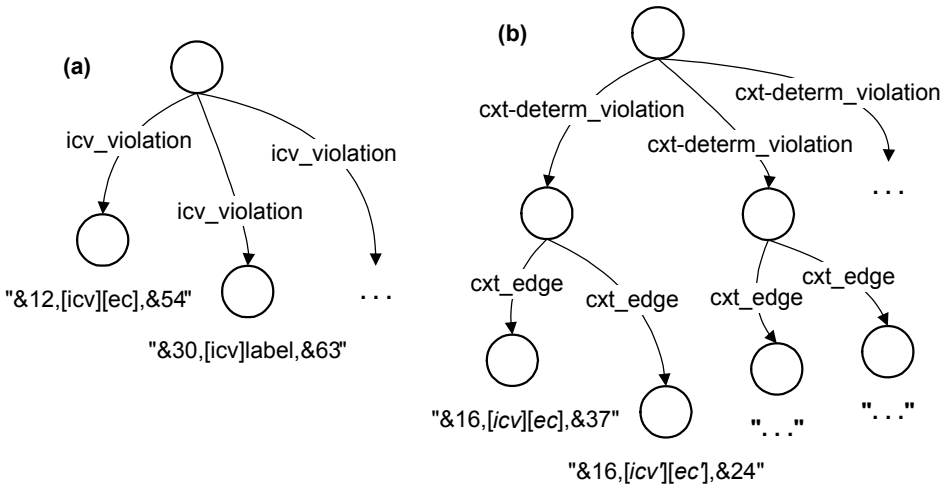
```
select cxt-determ_violation: {cxt_edge: @P, cxt_edge: @Q}
from root::# <X>,
     <X>(::[I][-])@P Y,
     <X>(::[J][-])@Q Z
where Y != Z
within [I] * [J] != [-]
```

The result of the query has the form depicted in Figure 5.4 (b)[86].

[85] In the case of `union`, as well as in `intersect` and `except` that perform the respective operations between queries, duplicate edges are considered to be edges that have the same label and that lead to objects corresponding to the same object in the database.

[86] Note that for every actual violation there will be two `cxt-determ_violation` objects in the result graph, corresponding to mirror bindings of `Y` and `Z`.

Figure 5.4: Finding empty inherited coverages and context-determinism violations in the database graph.



Label variables have the form `$X`, and bind to entity edge labels. A label variable cannot bind to context edge labels; in this case, labels are explicit contexts and context variables must be used. Label variables can be seen as a special case of path variables: `$X` could be expressed as `pathof(@X)`, with `@X` binding to a path that consists of a single entity edge. The benefit of having label variables as a separate variable type is that they can be used in the place of entity edge labels when constructing results[87]. Moreover, as pointed out in [ABS00], label variables can be used to "hide" specific subobjects of an object[88] in the results.

Consider the query:

```
select $L: <X>
from recreation_guide.-$L.name <X>
```

In the above query, label variable `$L` binds to the labels of entity edges that depart from *root* facets and lead through a context edge to `name` multidimensional objects. The variable binds to every entity edge label, because the wildcard – matches any entity edge label. The query returns `name` multidimensional objects, pointed by entity edges that denote the corresponding establishment type (music club, restaurant, etc.).

Label variables can also appear in the place of object values in `select`, converting database metadata to data in the results. The following MQL query

```
select distinct restaurant_attr: $L
from recreation_guide.restaurant.-$L
```

returns all different attributes of restaurants in the database of Figure 5.1, as object values:

[87] A path variable cannot be used to construct edges in the results, because a path string in the general case is not a legal label for an edge.

[88] Section 5.4.3.1 presents an example query that uses a context variable to "hide" facets of objects depending on explicit contexts (labels of context edges). Label variables can be used in a similar way to "hide" multidimensional objects depending on labels of entity edges.

```
{restaurant_attr: "name",
 restaurant_attr: "address",
 restaurant_attr: "review",
 restaurant_attr: "parking"}
```

Notice that MQL encloses the values of label variables in quotes when they are used as object values in the template of the `select` clause.

Summarizing, label variables bind to entity edge labels, and context variables when used as explicit context qualifiers bind to context edge labels (explicit contexts). On the other hand, a path variable can bind to entity edges, context edges, or any sequence of the two, but cannot be used as an edge label in the template of `select` for constructing results.

Label variables and path variables are usually attached to wildcards and gcpe_components, and bind to matching parts of the graph. However, a gcpe_component is not itself allowed to contain objects variables (enclosed in { }), path variables, label variables, or context variables used as explicit context qualifiers. Context path expressions that contain the above variables are called **augmented general context path expressions**, and their syntax is formally specified in Appendix A.

5.4.2 Nested MQL Queries

MQL queries can be nested. A subquery may appear in the `select` clause of a query, and new nodes created by the inner query are incorporated in the results of the outer query. Inner queries can use variables bound in outer queries, as in:

```
select club_comments: (select gr_comments: Y
                       from <X>.#.[% lang=gr]comments Y)
from recreation_guide.- <X>
```

In the above query, variable `<X>` binds to `music_club` and `restaurant` multidimensional objects. Then, for each such binding, the inner query gets the `comments` facets in Greek, irrespective of how deep they are in the database graph. The result is that the outer query groups matching `comments` facets according to the restaurant or music club they belong to (which, as mentioned in Section 5.3.2, does not happen automatically in MQL). When evaluated on the graph of Figure 5.1, the query returns:

```
{club_comments: {gr_comments: &10 "..."},
 club_comments: {gr_comments: &41 "...", gr_comments: &36 "..."}
}
```

The first line corresponds to `music_club`, while the second `club_comments` contains the comments for `restaurant`. Notice that if the inherited coverage qualifier `[% lang=gr]` did not exist in the query, then the second line of the results would also include node &42. Another interesting point is the use of the multidimensional object variable `<X>` instead of a context object variable `X`. The reason for this is that we want to group comments according to the multidimensional entity they refer to, not according to a particular club facet. In Figure 5.1 the `music_club` multidimensional entity consists of a single facet, but in the general case it could comprise several facets, each leading to a number of `comments` nodes.

For comparison, here is a similar query that does not group comments:

```
select gr_comments: X
from recreation_guide.#.[% lang=gr]comments X
```

The query gives:

```
{gr_comments: &10 "...",
 gr_comments: &41 "...",
 gr_comments: &36 "..."}
```

In Section 5.3.5 we mentioned that aggregate functions like `union([c])` can be used in the `context` clause; the relevant example of Section 5.3.5 returned the worlds under which there exists a `menu` facet for a music club with a specific oid. The following example elaborates on this, and returns the available menu languages for each restaurant and music club in the database.

```
select place: (select place_name: N, menu_langs: [Z]
               context [Z] as union([Y]) * [% lang in ALL]
               from <X>.menu::[Y][-],
                    <X>.name N)
from recreation_guide.- <X>
```

The above example uses a nested query to "group by" contexts before applying the aggregate function. Specifically, the outer query binds the variable `<X>` to music clubs and restaurants. Then, for each `music_club` or `restaurant` multidimensional object, the inner query binds the variable `N` to its `name` facets, and the context variable `[Y]` to the inherited coverage of context edges that lead to `menu` facets. The context union of those inherited coverages is computed in the `context` clause for the *current* music club or restaurant, using the aggregate function `union([c])`. The result of the aggregate function is a context specifier representing the worlds under which some menu facet holds for the specific place (music club or restaurant). This context specifier may include dimensions other than `lang`, like for example `detail` and `season`; to restrict the result to `lang`, we intersect the context specifier with the context pattern `[% lang in ALL]`, which "screens out" dimension specifiers that do not refer to `lang`. As already mentioned, `ALL` is shorthand for the complete domain of a dimension. So, `[Z]` denotes the languages for which there exists some world under which a menu facet holds. When evaluated on the graph of Figure 5.1, the query returns the mssd-expression:

```
{place: {place_name: &3 "...", menu_langs: [lang in {gr,en,fr}]},
 place: {place_name: &16 "..."}}
```

The first `place` object corresponds to the `music_club` of Figure 5.1. The second `place` object corresponds to the `restaurant` of Figure 5.1, and does not contain a `menu_langs` object because variable `[Y]` is bound to *nil*. To restrict the results to objects that do contain a menu, we could add the clause "`where exists <X>.menu::[-]`" to the query.

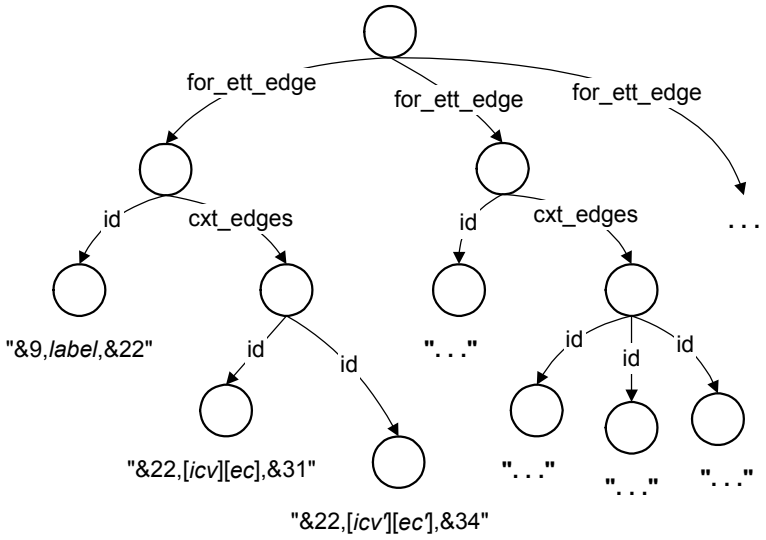
Observe that if a `name` multidimensional object of a music club or restaurant has many facets, the corresponding `place` object will contain as many (`place_name`, `menu_langs`) pairs. By using a multidimensional object variable `<N>` instead of `N`, we can avoid this repetition; however, pairs will still appear if a music club or restaurant contains many `name` multidimensional objects. Grouping the (hypothetically many) `name` objects of a place would require an additional nested query.

In MOEM, some facets of a multidimensional entity are accessible through an incoming entity edge, while others may not hold under a common world with the incoming edge. Therefore, depending on the name (entity edge label) we use for a multidimensional entity, different facets may be accessible under the corresponding worlds. The query that follows finds the facets that are not accessible by incoming entity edges, for each entity edge in the database.

```
select for_ett_edge: {id: @P, cxt_edges:
                                  (select id: @Q
                                   from <X>(::[J][-])@Q
                                   within [I] * [J] = [-])}
from root.#(.[I]-)@P <X>
```

The result of the query has the form of the graph in Figure 5.5.

Figure 5.5: Finding inaccessible facets for each incoming entity edge.



For each entity edge, object `id` specifies the edge, and object `cxt_edges` contains the `id`s of inaccessible context edges (leading to inaccessible context objects). If all facets of the multidimensional entity are accessible through an entity edge, the corresponding `cxt_edges` does not have any `id` children[89]. Observe how the context variable `[I]`, introduced in the outer query, is used in the inner query to determine whether an entity edge and a context edge do not hold under any common world.

5.4.3 Construction of Results

The result of an MQL query is a Multidimensional Data Graph in the form of an mssd-expression. MQL constructs results according to a template specified in the `select` clause. Due to the fact that Multidimensional Data Graph has two types of nodes and two types of edges, MQL must cover more possibilities than if results were conventional semistructured data.

In what follows, we first explain how the two distinct types of nodes and edges are handled by MQL when constructing results. We then discuss the use of context path expressions in the `select` clause. Finally, we discuss how MQL can **partially reduce** the

[89] In this case, the result graph will contain complex nodes as leaves. This is legal in MQL, where the result is a Multidimensional Data Graph. In Section 5.4.3.3 we explain the use of the keyword `holding` that eliminates such nodes.

result graph, returning a subgraph whose every node and edge actually holds under some world.

5.4.3.1 Constructing Entity Edges and Context Edges

While context path expressions are built around the canonical form of a Multidimensional Data Graph, the result graph of an MQL query needs not be in canonical form. So, when constructing results, MQL only needs to ensure that edges departing from a node are of the correct type: entity edges must depart from context nodes, and context edges must depart from multidimensional nodes. In MQL, results are constructed on the basis of an mssd-expression template in the `select` clause, which defines the sequence of context nodes and multidimensional nodes in the results. In this template, curly brackets { . . . } signify context objects, and angle brackets < . . . > signify multidimensional objects.

The template can contain the following in the place of an *entity edge* label:

- A string that comply with the entity edge label syntax. This string becomes the label of entity edges in the results.
- A label variable. Entity edges are created in the results having as labels the values bound to this label variable.

The template can contain the following in the place of a *context edge* label:

- A string that comply with the context specifier syntax. Context edges with this string as a label are created in the results.
- A context variable. Context edges are created in the results having as labels the context specifiers bound to this context variable.

Summarizing, the mssd-expression template can contain as edge labels: (a) strings and label variables within curly brackets (context objects), and (b) context specifiers and context variables within angle brackets (multidimensional objects). The template can contain any type of variable as object values: context object variables, multidimensional object variables, context variables, label variables, or path variables.

Especially for the root of the result graph, the – curly or angle – brackets can be omitted from the template. The type of the root node can be implied from the type of labels[90]. Consider the two equivalent queries:

```
select <[Y]: X>
from recreation_guide.#.comments::[Y][-] X

select [Y]: X
from recreation_guide.#.comments::[Y][-] X
```

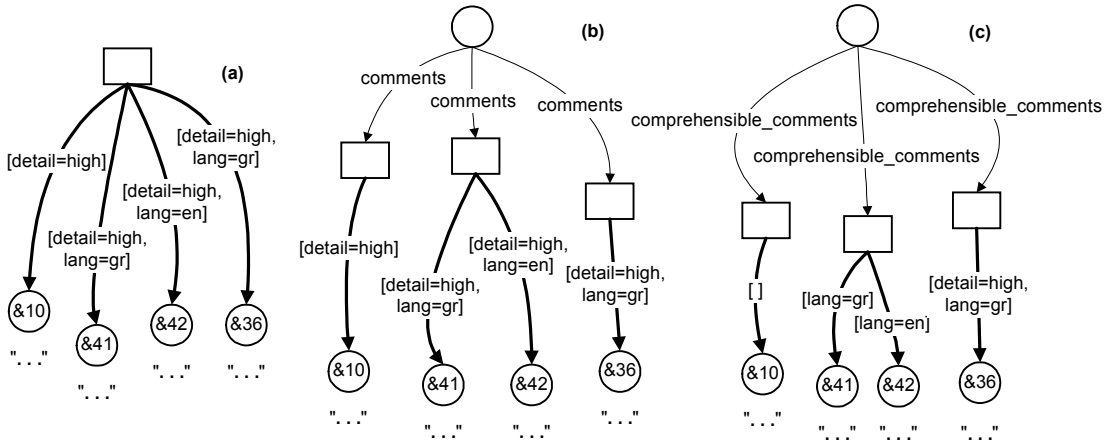
The effect of these queries is to "copy" all context edges that point to some `comments` facet from the database in Figure 5.1 to the results, with the difference that for each context edge the inherited coverage in the database becomes the explicit context in the results. All `comments` facets are gathered under one multidimensional object, which is the root of the result graph, and from which context edges depart:

[90] Obviously, if the template specifies more than one edge departing from the root, all edges must agree on the type of the root (context or multidimensional).

```
<[detail=high]: &10 "...",
 [detail=high,lang=gr]: &41 "...",
 [detail=high,lang=en]: &42 "...",
 [detail=high,lang=gr]: &36 "...">
```

The fact that the root is a multidimensional object is explicitly declared in the first query with the two angle brackets < and > that confine the mssd-expression template in the `select` clause. In the second query, the same fact is deduced from the context variable `[Y]` that appears in the place of *the first edge label*. The results are depicted as a graph in Figure 5.6 (a).

Figure 5.6: Results of MQL queries.



The following nested query refines the example above and retains the original grouping of `comments` facets under their respective multidimensional objects. The curly brackets in the outer query and the angle brackets in the inner query that correspond to the root can be omitted. Figure 5.6 (b) shows the results pictorially.

```
select {comments: (select <[Y]: Z>
                   from <X>::[Y][-] Z)}
from recreation_guide.#.comments <X>
```

The result of the query in textual form is:

```
{comments: <[detail=high]: &10 "...">,
 comments: <[detail=high,lang=gr]: &41 "...",
            [detail=high,lang=en]: &42 "...">,
 comments: <[detail=high,lang=gr]: &36 "...">}
```

As another example consider the following query, which uses a context variable to "hide" from the results all `comments` facets that are not in one of the four languages understood by the user. The filtering of incomprehensible comments is done in the `within` clause. A detail is that in this case context variable `[Y]` binds to explicit contexts, rather than inherited coverages.

```
select comprehensible_comments:
               (select [Y]: Z
                from <X>::[Y] Z
                within [Y] * [lang in {gr,en,fr,sp}] != [-])
from recreation_guide.#.comments <X>
```

The graph of the results is given in Figure 5.6 (c). For each multidimensional `comments` object bound to `<X>`, an entity edge labeled `comprehensible_comments` is created that points to a multidimensional node. The multidimensional node is the root of the inner query, and corresponds to the node that is the current value of `<X>`, except from the fact that only comment facets in Greek, English, French, and Spanish are copied from the database to the results.

5.4.3.2 Context Path Expressions in "select"

Context path expressions can appear in the `select` clause. The query

```
select gr_comments: recreation_guide.#.[% lang=gr]comments::[-]
```

is equivalent to the query

```
select gr_comments: X
from recreation_guide.#.[% lang=gr]comments X
```

which, as we have seen in Section 5.4.2, returns all `comments` facets in Greek in the database:

```
{gr_comments: &10 "...",
 gr_comments: &41 "...",
 gr_comments: &36 "..."}
```

As shown by the two equivalent queries, if a context path expression appears in the `select` clause then the `from` clause may be inferred, exactly like in Lorel [AQM+97].

A context path expression in `select` must end with a facet part (`::[-]` in the case of the query above) to cause evaluation to context objects. If the context path expression ends with an entity part (`.[% lang=gr]comments` in the query that follows), then the results of the query will contain multidimensional objects, as if a multidimensional object variable had been used:

```
select comments: recreation_guide.#.[% lang=gr]comments

select comments: <X>
from recreation_guide.#.[% lang=gr]comments <X>

{comments: &100 <...>,
 comments: &27 <...>,
 comments: &35 <...>}
```

Oid &100 is used above to denote the multidimensional node whose only facet is node &10, and which is introduced by the transformation of the graph in Figure 5.1 to canonical form.

Inferring Edge Labels

Edge labels can be omitted from the mssd-expression template in `select`. In this case, two issues arise: (a) how to infer the missing edge labels in the results, and (b) how to determine the type of the root node. Concerning issue (b), as we have mentioned in Section

5.4.3.1, if brackets are not there to indicate the type of the root, MQL looks at the type of the first edge label. Therefore, it is not allowed to omit brackets *and* edge labels at the same time, and if labels are missing *brackets must be present*. We continue with issue (a).

The following two queries are very similar to the two preceding queries, but edge labels are missing from the mssd-expression template. The first query returns `comments` facets in Greek, while the second returns `comments` multidimensional objects that comprise some facet in Greek:

```
select {recreation_guide.#.[% lang=gr]comments::[-]}

select {recreation_guide.#.[% lang=gr]comments}
```

In both queries, context path expressions must be enclosed in curly brackets, to declare that the root of the result graph is a context object. Curly brackets cause *entity* edges to be created, labeled after the last entity edge in the matching context data path that leads to the respective node in the database. In the case at hand, this entity edge is always labeled `comments`, however if a context path expression contains wildcards or regular expressions, then the label in the results will be the same as the label in the database of the edge leading to the corresponding object. This can be viewed as if a label variable had been used:

```
select $L: OBJ_VAR
from cxt_path_expr$L OBJ_VAR
```

If a missing entity edge label cannot be inferred[91], the default label `answer` is used when constructing the result graph.

It is possible for the root of the results to be a multidimensional object. This causes context edges to be created, labeled like the *final* context edge in the matching context data path that leads to the respective node in the database. In effect, this means that explicit contexts are "copied" in the results paired with the facets they qualify in the database:

```
select <recreation_guide.#.[% lang=gr]comments::[-]>
```

This query is equivalent to:

```
select [Y]: X
from recreation_guide.#.[% lang=gr]comments::[Y] X
```

Evaluated on the graph of Figure 5.1 the above query returns:

```
<[]: &10 "...",
 [lang=gr]: &41 "...",
 [detail=high,lang=gr]: &36 "...">
```

In order to infer context edge labels the final edge of the corresponding context data path *must* be a context edge. In case a context edge label cannot be inferred, then it defaults to the empty context `[-]`, which is used as the explicit context of context edges in the results.

Although this section introduces omission of edge labels under the pretext of context path expressions in `select`, edge labels may be omitted irrespective of whether or not context path expressions exist in `select`; rules for inferring missing edge labels are in any case the same.

[91] For instance, an entity edge label cannot be inferred if it is the label of a user-defined object in the template of the `select` clause, which may contain more than one variable.

Grouping Results

As we have seen in previous sections, nested queries can be used to group results. An alternative way to group results is to use context path expressions in the `select` clause "bound" to a `from` clause:

```
select club_comments:
         {gr_comments: <X>.#.[% lang=gr]comments::[-]}
from recreation_guide.- <X>

{club_comments: {gr_comments: &10 "..."},
 club_comments: {gr_comments: &41 "...", gr_comments: &36 "..."}}
```

The context path expression in `select` can be interpreted as a second `from` clause, which is part of an implied nested query. The above query is equivalent to the first nested query example of Section 5.4.2, and returns comment facets in Greek grouped by establishment (music club or restaurant).

It is possible to omit labels in front of context path expressions, in which case they are inferred as described in the previous section:

```
select club_comments: {<X>.#.[% lang=gr]comments::[-]}
from recreation_guide.- <X>
```

An equivalent nested query is:

```
select club_comments: (select {Y}
                       from <X>.#.[% lang=gr]comments Y)
from recreation_guide.- <X>
```

The results are the same as before, with the difference that the inferred `comments` label is used instead of the `gr_comments` label. Observe that the two equivalent queries above use context path expressions that conclude to different parts. When context path expressions are used in `select`, context object variables cannot be used to force evaluation to context objects. Therefore, context path expressions in `select` must conclude to the proper part: entity part for evaluation to multidimensional objects, and facet part for evaluation to context objects.

A context path expression in `select` may not always be viewed as an inner query. Compare the following two queries and their results.

```
select {{restaur_name: <X>.name::[-], winter_floor: Y}}
from recreation_guide.restaurant <X>,
     <X>.[season=winter]address.floor Y

{answer: {restaur_name: ...,
          restaur_name: ...,
          winter_floor: ...},
 answer: ...
}

select {{restaur_name: {<X>.name::[-]}, winter_floor: Y}}
from recreation_guide.restaurant <X>,
     <X>.[season=winter]address.floor Y
```

```
{answer: {restaur_name: {name: ...,
                         name: ...},
          winter_floor: ...},
 answer: ...
}
```

The queries return for each restaurant its name facets and the floor it operates in winter. The outer curly brackets imply a missing entity edge label (which defaults to `answer`), and keep together the restaurant names and the `winter_floor` of each restaurant under a common parent object. The only difference is that in the first query the context path expression `<X>.name::[-]` is *not* enclosed in brackets, while in the second query it is enclosed in brackets.

In the first query, the context path expression is not equivalent to an inner query, in the sense that it does not return a graph with a single root for each binding of `<X>`. Instead, two `restaur_name` edges point to two name facets of a restaurant. The second query encloses the context path expression in brackets, which can be seen as *denoting the root type of an equivalent inner query* (curly for context, angle for multidimensional). As a result, for each restaurant a `restaur_name` edge points to an object that can be considered the root of the implied inner query, and is the parent of `name` facets.

An important point is that a context path expression in `select` is always treated by MQL in the same way; whether or not the same result could have been achieved with an inner query, depends on the grouping of brackets in the template of the `select` clause. The semantics of a context path expression in `select` is explained below:

```
select ... label: X.cpe2
from cpe1 X

select ... label: Y_X
from cpe1 X, X.cpe2 Y_X
```

Variable Y_X evaluates to a set for every value of `X`. If `label` is not attached to the root of the results, MQL creates a node for every value of `X`. Then, `label` edges depart from each such node, leading to the values of the corresponding Y_X. On the other hand, if `label` is attached to the root of the results, separate nodes for values of `X` cannot be created and the values of all the sets bound to Y_X are grouped under the root.

5.4.3.3 Reducing the MQL Result Graph

Irrespective of whether the database is an MOEM or not, the graph returned by an MQL query may contain nodes and edges that do not hold under *any* world. We often want to ensure that every node and edge in the results holds under at least one world. The keyword `holding` follows the keyword `select` in the `select` clause, and causes the removal of all nodes and edges that have empty inherited coverage. As shown in the previous chapter, the result of this process is a Multidimensional Data Graph whose leaves are exclusively atomic nodes, and which contains nodes and edges that "survive" as part of a conventional OEM holding under some world. In case the result graph of an MQL query is context-deterministic, using `holding` guarantees that the query will return an MOEM.

The process is the last operation that takes place before returning the results of a query, and is applied to the complete graph that is constructed based on the mssd-expression template in `select`, after template variables have been substituted by bound values and inner queries have been evaluated. In order to remove nodes and edges that do not hold under any world, inherited coverages of the result graph need to be calculated.

In effect, `holding` causes what we have described in the previous chapter as **graph context pruning**. As we have seen in the previous chapter, the combination of **graph context projection** to a context c together with graph context pruning, is equivalent to **partial reduction** for context c. Consider the query:

```
select holding guide_facets:
               <[detail=low]: <X>, [detail=high]: <X>>
from recreation_guide <X>
```

The variable `<X>` binds to the root of the database, and in the template it is pointed to by a couple of context edges with explicit contexts `[detail=low]` and `[detail=high]`. Those context edges actually project the database to their respective contexts; then the keyword `holding` causes nodes and edges with empty inherited coverage to be removed. The result is a graph that comprises two partial reductions of the database, one for the context `[detail=low]` and another for the context `[detail=high]`.

The following query returns a graph that has the following structure: the root is a multidimensional node, and departing context edges lead to all possible facets of the database that hold under some world.

```
select holding <[Z]: <X>>
context [Z] as extension([Y])
from [Y]recreation_guide <X>
```

The variable `<X>` binds to the root of the database, and the context variable `[Y]` binds to the inherited coverage of the root. Remember that the inherited coverage of the root represents exactly the worlds under which the graph can be reduced to OEMs. The context variable `[Z]` binds to those worlds through the function `extension([c])`, which, as mentioned in Section 5.3.5, turns a context into the worlds it represents. The effect of using the context variable `[Z]` in `select` is that a new context edge is constructed for every world under which the database can be reduced to OEM[92]. The context edges have explicit contexts that correspond to those worlds, projecting separately the database to each of those worlds.

From the previous chapter, **reduction to OEM** under a world w is equivalent with graph context projection to a context representing only w, graph context pruning, and **graph de-contextualization**. Consequently, the result of the query above is a graph comprising all separate OEM facets of the database, with the difference that they have not been de-contextualized. This means that multidimensional nodes and context edges, although not significant any more, have not been removed, leaving OEM facets in an MOEM form.

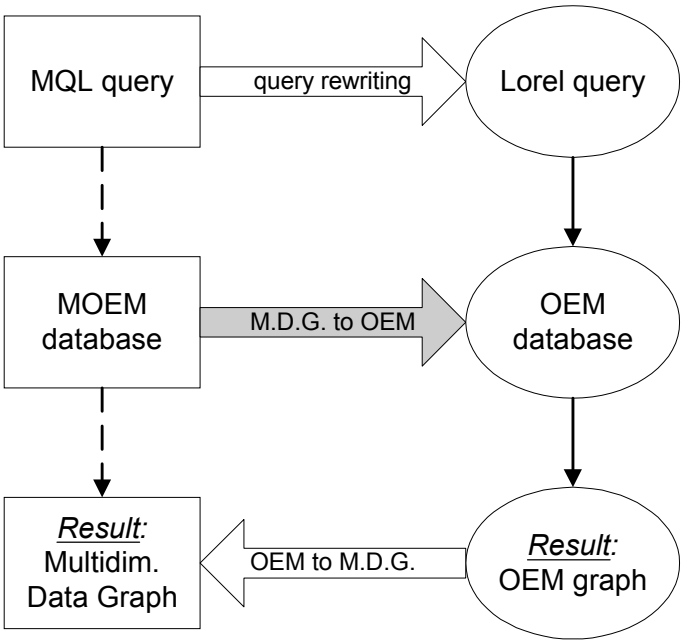
5.5 IMPLEMENTATION ON TOP OF LORE

Analogously to Lorel, which has been implemented on top of an object database [AQM+97], we have implemented MQL on top of LORE [MAG+97]. Our prototype MQL implementation consists of an application that is initialized with an MOEM database, receives MQL queries, and returns Multidimensional Data Graphs that are the results of these queries. The application interfaces with LORE, which is used as a back-end. We have chosen LORE as a basis for implementing MQL, because it is interesting to see: (a) how an MQL query

[92] For the query to have this effect, the inherited coverage of the root must not contain the universal clause ∅. As explained previously, the **context extension** of a context `[c]` is not defined if `[c]` contains the universal clause ∅, and `extension([c])` returns *nil*. In this case no context edges are created, and the result before `holding` is applied consists only of a multidimensional root node.

compares with an "equivalent" Lorel query, and (b) how an MOEM can be expressed through a conventional OEM.

Figure 5.7: Evaluating MQL queries using the LORE infrastructure.



The overall architecture is shown in Figure 5.7. The process we want to implement is depicted as a dashed line, which starts from an MQL query, passes through an MOEM database, and concludes with a Multidimensional Data Graph that is the result of the query.

The process that actually takes place is depicted as a normal line, and shows a Lorel query evaluated on an OEM database that returns an OEM graph as a result. This line together with the ellipse-shaped boxes is part of LORE, which is controlled by our application through the programming interface that it provides.

The main issue is to define a transformation T from Multidimensional Data Graphs M to OEMs O = T(M), with the following properties: (a) the reverse transformation T^{-1} exists, and if O is given then M = T^{-1}(O) can be recovered, and (b) it is possible to translate an MQL query q_M to an "equivalent" Lorel query q_L. By equivalent we mean that if q_M evaluated on M returns M′ and q_L evaluated on O returns O′, then T(M′) = O′. Then, the answer to q_M can be computed by evaluating q_L on T(M), and by applying the reverse transformation T^{-1}(O′) to the results of q_L.

Those transformations and the MQL query translation are depicted in Figure 5.7 as thick horizontal arrows. The application, among other things, implements those arrows and performs the following key steps:

1. Converts an MOEM database to an OEM database, which becomes the database of LORE.
2. Translates an MQL query to a Lorel query, which is passed over to LORE for evaluation on the OEM database.

3. Gets the results from LORE, and converts them from OEM back to Multidimensional Data Graph.

The Multidimensional Data Graph of step 3 is the result of the MQL query of step 2 evaluated on the MOEM database of step 1. Step 1 initializes the database and corresponds to the gray horizontal arrow of Figure 5.7, while steps 2 and 3 are carried out every time an MQL query is submitted.

In what follows, we focus on specifying the three key steps listed above. The actual application that implements them and evaluates MQL queries is part of a more comprehensive system for MSSD, which is presented in Chapter 8.

5.5.1 Transforming MOEM Databases to OEM

In order to transform MOEM to OEM, we must use OEM elements to represent MOEM elements that do not have a counterpart in OEM, namely context edges and multidimensional nodes: context edges can be represented by OEM edges that have some special label, while multidimensional nodes correspond to OEM nodes from which these special edges depart. We must also find a way to encode context in OEM. Context cannot just appear as a string whose value is a context specifier. The reason is that Lorel does not understand the notion of context, thus if context-driven queries are to be translated to Lorel queries, context must be encoded in a way Lorel can understand and handle. Contexts that must be encoded include explicit contexts, but also inherited coverages of edges, which play an important role in context path expressions and MQL queries.

Figure 5.8: Representing Multidimensional Data Graph using OEM.

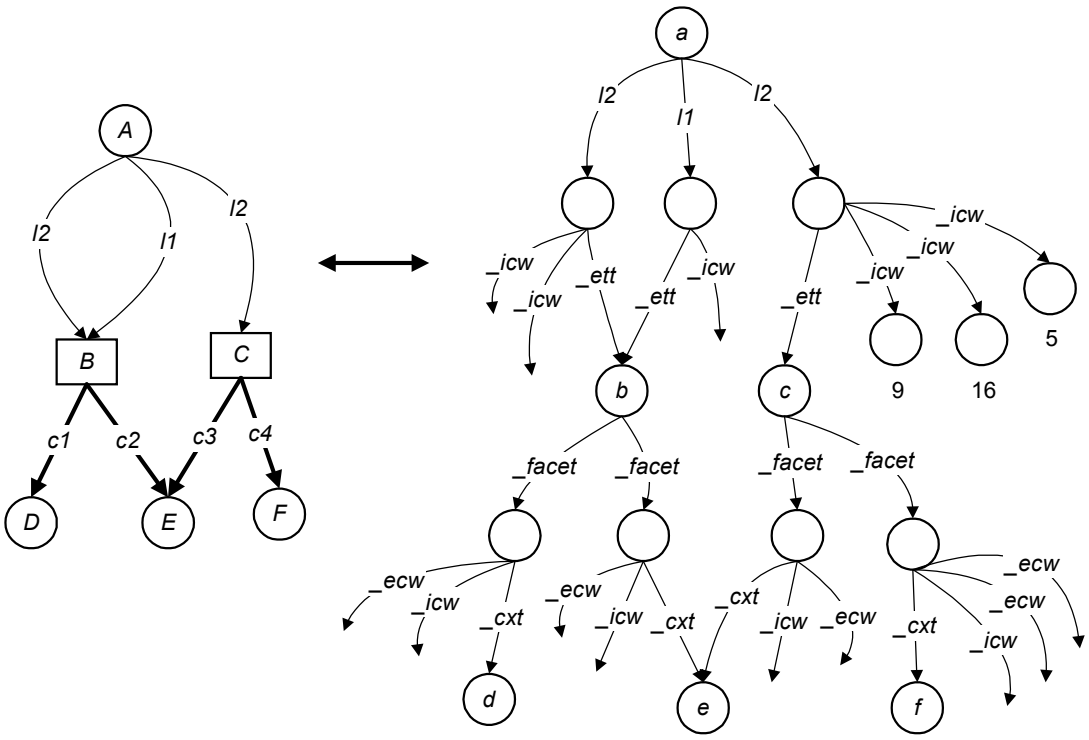


Figure 5.8 gives an intuition of our transformation. It presents a simple Multidimensional Data Graph M together with its OEM counterpart O. Nodes with a capital letter in M correspond to nodes with the respective lowercase letter in O. Observe that for each edge in M an additional node exists in O, splitting the edge in two OEM edges. The role of this node is to group the encoded context(s) for the corresponding Multidimensional Data Graph edge. A number of reserved labels with special meaning are also used. All reserved labels start with an underscore, and they are: _ett, _facet, _cxt, _icw, and _ecw. An entity edge is represented by an edge with the same label and a following edge labeled _ett. A context edge is represented by an edge labeled _facet and a following edge labeled _cxt. Explicit contexts and inherited coverages of edges are converted to the worlds they represent, and all possible worlds are mapped to integers[93]. Edges that are labeled _icw point to enumerated worlds (integers) that belong to inherited coverages, while edges labeled _ecw point to enumerated worlds (integers) that belong to explicit contexts.

The actual transformation process of a Multidimensional Data Graph $M = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ to an OEM O is given below. The process assumes that there exists a mapping from all possible worlds to integers.

O ← MDGToOEM (M) is:

1. For every world, add a new atomic node w to V_{cxt} that corresponds to that world, having as value the respective integer.
2. Move all nodes from V_{mld} to V_{cxt} as complex nodes.
3. For every edge $h = (q, l, p) \in E_{ett}$ add a new complex node u to V_{cxt}. Then, replace h with the new edges (q, l, u) and (u, _ett, p) in E_{ett}. Next, for every world represented by the inherited coverage of h, add an edge (u, _icw, w) to E_{ett}, where w corresponds to that world.
4. For every edge $h = (q, c, p) \in E_{cxt}$ add a new complex node u to V_{cxt}. Then, remove h from E_{cxt}, and add the edges (q, _facet, u) and (u, _cxt, p) to E_{ett}. Next, for every world represented by the inherited coverage of h, add an edge (u, _icw, w) to E_{ett}, where w corresponds to that world. Moreover, for every world represented by the explicit context of h, add an edge (u, _ecw, w) to E_{ett}, where w corresponds to that world.
5. Return $O = (V_{cxt}, E_{ett}, r, v)$.

Some atomic nodes that correspond to worlds may not be pointed to by any edge, and be unreachable from the root. This can be easily avoided by adding those nodes to V_{cxt} at the moment they are used in steps 3 and 4 instead of step 1. Alternatively, unreachable nodes can be removed at the end of the process. We leave this to implementation. In addition, the function v should be updated to include the values for the new nodes.

Notice that, if the Multidimensional Data Graph has complex nodes or multidimensional nodes as leaves, then the resulting OEM will also have complex nodes as leaves. Actually, in step 2 we could represent complex leaves of Multidimensional Data Graph as OEM atomic nodes that have the reserved value "_C", and multidimensional leaves as OEM atomic nodes that have the reserved value "_M". This would guarantee that any Multidimensional Data Graph can be transformed to a valid OEM, and that in the reverse transformation it will be

[93] Therefore, this encoding of context requires that the set of dimensions **D** is known beforehand, in contrast to MOEM and MQL that do not use **D** to represent and query MSSD.

possible to identify leaf nodes for what they are. However, we are mainly interested in transforming MOEM databases, where leaves are exclusively atomic nodes.

5.5.2 Translating MQL Queries to Lorel

MQL queries can be translated to "equivalent" Lorel queries, which are evaluated on the OEM given by the transformation defined in the previous section. For this translation to work, the MOEM database *must be in canonical form* when the transformation to OEM takes place. This allows context path expressions, which are built around the canonical form, to be translated to "equivalent" conventional path expressions.

In this section we specify such a translation that supports the major features of MQL, in order to get an impression of how an equivalent Lorel query would look like. We have not addressed some features of MQL that seemed difficult or impossible to translate, like regular expressions in context path expressions (general context path expressions). We have also left out MQL features that did not seem essential for our purpose, like for example the `context` clause and context patterns.

Moreover, we do not specify a translation for the `select` clause of MQL, because the `select` clauses in MQL and Lorel are very similar semantically. In spite of their similarity, the `select` clauses of the two languages differ in defining the graph that variable bindings hung from. Consequently, given that the `select` clause of MQL is not translated, the returned graph will be constructed according to the Lorel interpretation of the MQL `select` clause. This, however, is not prohibitive for simple demonstrative queries.

5.5.2.1 Context Path Expressions to Path Expressions

To facilitate the comprehension of translated Lorel queries, we use `E-HUB` as identifier for nodes from which a _ett edge departs, `MLD` for nodes from which a _facet edge departs, and `C-HUB` for nodes from which a _cxt edge departs. In addition, we use w_1, w_2, ... to denote the integers that have been mapped to worlds.

We start with a very simple MQL `from` clause:

```
from X.[c]label Y
```

We assume `[c]` is a context specifier representing the worlds that correspond to w_4, w_7, and w_9. From Proposition 5.1 and because `[c]` is a context specifier, it can be repeated throughout the path it qualifies, and the clause can be written as:

```
from X.[c]label::[c][-] Y
```

This `from` clause can also be written in MQL as:

```
from X.[c]label <V>,
     <V>::[c][-] Y
```

The equivalent Lorel expression is:

```
from X.label E-HUB,
     E-HUB._ett MLD,
     MLD._facet C-HUB,
     C-HUB._cxt Y
where     E-HUB._icw{W4} = w4
      and E-HUB._icw{W7} = w7
      and E-HUB._icw{W9} = w9
      and C-HUB._icw{W4} = w4
      and C-HUB._icw{W7} = w7
      and C-HUB._icw{W9} = w9
```

The `where` clause states that the inherited coverages of the two MOEM edges must contain all the worlds specified by `[c]`. Observe the use of the variables `W4`, `W7`, and `W9`, which declare that it is not the *same* node that must be equal to w_4, to w_7, and to w_9 (otherwise the condition would always be false)[94].

We now consider a complete MQL query:

```
select restaurant: {name: P, winter_floor: Y}
from recreation_guide.restaurant X,
     X.[season=winter]address.floor Y,
     X.[season=summer,daytime=noon]address.floor Z,
     X.name P
where Z="terrace"
```

For brevity, we use `[c1]` to denote the context specifier `[season=winter]`, and `[c2]` to denote `[season=summer,daytime=noon]`. The MQL query can now be written as:

```
select restaurant: {name: P, winter_floor: Y}
from [-]recreation_guide <V1>, <V1>::[-][-] V2,
     V2.[-]restaurant <V3>,    <V3>::[-][-] X,
     X.[c1]address <V4>,       <V4>::[c1][-] V5,
     V5.[c1]floor <V6>,        <V6>::[c1][-] Y,
     X.[c2]address <V7>,       <V7>::[c2][-] V8,
     V8.[c2]floor <V9>,        <V9>::[c2][-] Z,
     X.[-]name <V10>,          <V10>::[-][-] P
where Z="terrace"
```

The equivalent Lorel query is:

```
select restaurant: {name: P, winter_floor: Y}
from
     recreation_guide E-HUB1, E-HUB1._ett V1,
     V1._facet C-HUB1, C-HUB1._cxt V2,

     V2.restaurant E-HUB2, E-HUB2._ett V3,
     V3._facet C-HUB2, C-HUB2._cxt X,

     X.address E-HUB3, E-HUB3._ett V4,
     V4._facet C-HUB3, C-HUB3._cxt V5,

     V5.floor E-HUB4, E-HUB4._ett V6,
     V6._facet C-HUB4, C-HUB4._cxt Y,
```

[94] When using variables in this way, Lorel implies the condition:
`and W4 <> W7 and W4 <> W9 and W7 <> W9`

```
      X.address E-HUB5, E-HUB5._ett V7,
      V7._facet C-HUB5, C-HUB5._cxt V8,

      V8.floor E-HUB6, E-HUB6._ett V9,
      V9._facet C-HUB6, C-HUB6._cxt Z,

      X.name E-HUB7, E-HUB7._ett V10,
      V10._facet C-HUB7, C-HUB7._cxt P
where
      Z="terrace"
      and predicate(E-HUB3)
      and predicate(C-HUB3)
      and predicate(E-HUB4)
      and predicate(C-HUB4)
      and predicate(E-HUB5)
      and predicate(C-HUB5)
      and predicate(E-HUB6)
      and predicate(C-HUB6)
```

The expressions *predicate*(VAR) ensure that the corresponding edges have a proper inherited coverage. Therefore, each expression *predicate*(VAR) must be replaced by

```
          VAR._icw{W1} = w1
      and VAR._icw{W2} = w2
      and VAR._icw{W3} = w3
      and ...
```

where w_1, w_2, w_3, ... are the integers that correspond to worlds of the respective inherited coverage qualifier: for E–HUB3, C–HUB3, E–HUB4, and C–HUB4 the inherited coverage qualifier is [season=winter], while for E–HUB5, C–HUB5, E–HUB6, and C–HUB6 the inherited coverage qualifier is [season=summer,daytime=noon]. The edges that correspond to variables E–HUB1, C–HUB1, E–HUB2, C–HUB2, E–HUB7, and C–HUB7 can have any inherited coverage because their inherited coverage qualifier is the empty context [-], thus they are not included in where.

Using the above framework, it is straightforward to translate MQL queries that contain multidimensional object variables. Actually, the analytical form of our MQL query example contains the multidimensional object variables <V1>, <V3>, <V4>, <V6>, <V7>, <V9>, and <V10>, which correspond to the variables V1, V3, V4, V6, V7, V9, and V10 of the equivalent Lorel query. In addition, it is easy to accommodate explicit context qualifiers. A facet part ::[c_I] [c_E] will result in a predicate of the form:

```
          VAR._icw{W1} = w1
      and VAR._icw{W2} = w2
      and VAR._icw{W3} = w3
      and ...
      and VAR._ecw{W2} = w2
      and VAR._ecw{W6} = w6
      and ...
```

where w_1, w_2, w_3, ... correspond to the worlds of the inherited coverage qualifier [c_I], and w_2, w_6, ... correspond to the worlds of the explicit context qualifier [c_E].

Although we can translate the wildcard – of MQL to the wildcard % of Lorel, it is not always possible to translate MQL queries that contain the wildcard #. Specifically, if the inherited coverage qualifier of the path is not an empty context, it is not possible to translate MQL queries with regular expressions that use + or ∗. The reason is that we cannot introduce

variables throughout the (unknown) path so as to assert conditions about the corresponding inherited coverages in `where`.

5.5.2.2 Context Variables and "within" Clause

Consider the MQL query

```
select comments: Y
from recreation_guide.restaurant.[X]review.comments Y
within [X] * [detail=high] <= [lang=gr]
```

which uses a context variable to find comment facets in Greek in high detail[95]. The first step is to express context specifiers as Lorel queries. Suppose that `[detail=high]` represents the worlds that correspond to w_1, w_2, and w_3. The following Lorel query can then be used to express `[detail=high]`:

```
select W
from recreation_guide.#._icw W
where W=w1 or W=w2 or W=w3
```

Lets use $L_{[detail=high]}$ to refer to this query, and $L_{[lang=gr]}$ to refer to the similar Lorel query that expresses `[lang=gr]`. In addition, we use the symbol L_{CXT_VAR} to refer to a Lorel query that expresses the path inherited coverage of `review::[-].comments::[-]`, which is the value of the context variable `[X]`. This Lorel query is:

```
E-HUB3._icw intersect C-HUB3._icw intersect
E-HUB4._icw intersect C-HUB4._icw
```

The query evaluates to the "worlds" under which all edges of the path hold. Now that we have expressed all contexts as queries evaluating to sets of nodes that represent worlds, we can express context subset as a relation between the queries. Assuming that *query1* expresses a context `[c1]` and *query2* a context `[c2]`, the condition `[c1] <= [c2]` (`[c1]` context subset of `[c2]`) is implemented by the predicate

```
for all LEFT in (query1):
exists RIGHT in (query2): LEFT = RIGHT
```

where `LEFT` and `RIGHT` are variables that range over the "worlds" to the left and to the right side of the symbol `<=`, respectively.

The MQL query can now be translated to the following Lorel query:

```
select comments: Y
from
     recreation_guide E-HUB1, E-HUB1._ett V1,
     V1._facet C-HUB1, C-HUB1._cxt V2,

     V2.restaurant E-HUB2, E-HUB2._ett V3,
     V3._facet C-HUB2, C-HUB2._cxt V4,

     V4.review E-HUB3, E-HUB3._ett V5,
     V5._facet C-HUB3, C-HUB3._cxt V6,
```

[95] The query can be expressed in a simpler way, but we use that one as a more general case for the sake of this example.

```
      V6.comments E-HUB4, E-HUB4._ett V7,
      V7._facet C-HUB4, C-HUB4._cxt Y
where
       for all LEFT in (L[CXT_VAR] intersect L[detail=high]):
       exists RIGHT in (L[lang=gr]): LEFT = RIGHT
```

Using Lorel queries in a similar way, it is possible to implement all context operations in the `within` clause.

5.5.3 Transforming OEM Results to Multidimensional Data Graph

LORE returns the result of a Lorel query as an OEM graph[96]. As already explained, this OEM graph can be transformed to a Multidimensional Data Graph, which is the result of the original MQL query.

The process that transforms an OEM O = (V, E, r, v) to a Multidimensional Data Graph M is given below.

M ← OEMToMDG (O) is:

1. Represent O as a Multidimensional Data Graph M = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v), where V_{cxt} = V, and E_{ett} = E, and V_{mld}, E_{cxt} are empty sets.
2. For every edge h = (q, l, u) ∈ E_{ett} where l is not a reserved label, remove u from V_{cxt}. Then remove h and (u, _ett, p) from E_{ett}, and add the edge (q, l, p) to E_{ett}. Remove all edges (u, _icw, w) from E_{ett}.
3. For every edge h = (q, _facet, u) ∈ E_{ett}, move q from V_{cxt} to V_{mld} (if not already moved), and remove u from V_{cxt}. For all nodes w, where (u, _ecw, w) ∈ E_{ett}, apply context union to the corresponding worlds to get a context specifier c. Then remove h and (u, _cxt, p) from E_{ett}, and add the edge (q, c, p) to E_{cxt}. Remove all edges (u, _ecw, w) and (u, _icw, w) from E_{ett}.
4. Remove from V_{cxt} all nodes that correspond to worlds.
5. Return M = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v).

Step 4 removes all the atomic nodes with integer values that correspond to possible worlds. Those nodes can be easily spotted because at that moment they are unreachable from the root r. At the end of the process, function v may include more value assignments to nodes than actually exist. Notice that, in order to reconstruct context specifiers, step 3 needs the *same* mapping of worlds to integers that was used while transforming the MOEM database to OEM.

5.6 SUMMARY

In this chapter we presented **Multidimensional Query Language**, or **MQL** for short, a query language for multidimensional semistructured data. The focus of MQL is *context-driven queries*: queries where context must be taken into account for choosing the right data.

[96] Encoded in XML.

MQL is largely based on Lorel and retains its major contributions, namely path expressions and coercion. In addition, it introduces a number of new features in order to:

- *Manipulate context in queries*. Context in MQL is recognized as such and treated as first-class citizen. To directly support context conditions and context operations, MQL uses additional clauses.
- *Handle a more complex data model*. The data model of MQL is Multidimensional Data Graph, which uses two types of nodes and two types of edges for representing multidimensional information entities and their facets. This calls for a new way to navigate and define access patterns for such graphs. Moreover, it affects the way result graphs are constructed.

To address those issues we defined **context path expressions**, which incorporate context in conventional path expressions, and employ the canonical form of Multidimensional Data Graphs to formulate context-aware access patterns. Context path expressions use context specifiers, **context variables**, and **context patterns** to pose context conditions on the database graph. We enhanced context path expressions by integrating regular expressions in a way that allows only meaningful combinations of edge types. An interesting point is that a class of context path expressions can be viewed as defining joins between conventional OEMs that hold under different worlds.

Next, we introduced MQL, with two special clauses for handling context: clause `within` expresses conditions on context, and clause `context` creates new context variables to be used in the construction of results. We examined the `select` clause in detail; we focused on the construction of results that consist of two types of nodes and two types of edges, and on the use of context path expressions in `select`. Then, we explained how MQL reduces the result graph, making use of properties presented in the previous chapter. To complete the discussion on MQL, a number of features that are not particularly affected by context were presented, like path variables, label variables, and nested queries. In Appendix A we formally define the syntax of context path expressions and MQL.

Our prototype implementation of MQL demonstrates how an MQL query compares to an equivalent Lorel query: in context-driven queries, MQL and MOEM are obviously far more expressive and elegant than Lorel and OEM, while in conventional queries MQL and context path expressions become as simple to formulate as Lorel and conventional path expressions.

In our view, the potential of MSSD and the expressiveness of MQL justify the extensions to OEM we introduced in the previous chapter. Those additional element types of Multidimensional Data Graph are used by MQL to formulate *cross-world queries*, which have no counterpart in context-unaware databases.

6 USING MSSD TO ACCOMMODATE CHANGES

Multidimensional semistructured data (MSSD) is semistructured data that present different facets under different contexts. Various aspects of MSSD have been explicated in the previous chapters. Context has been defined as representing alternative worlds, and is expressed by assigning values to a set of user-defined variables called dimensions. The notion of context has been incorporated in OEM, leading to Multidimensional OEM (MOEM), a graph model for MSSD. Multidimensional Query Language (MQL) has been proposed for posing context-driven queries on MSSD.

In this chapter, we use what we have developed in previous chapters to give a solution to a specific problem, and present an example application in order to demonstrate the potential of MSSD. The problem can be stated as follows: given an OEM graph that comprises the database, we would like a way to represent dynamically changes in this database as they occur, keeping a history of transitions, so that we are able to subsequently query on those changes.

In what follows, we start by reminding the definitions of the four OEM basic change operations, and by introducing our own **basic change operations** for MOEM. We then define the way MOEM can be used to represent the *history of an OEM database*. Next, we discuss how Multidimensional Data Graph properties are applied in the case of representing OEM histories, and show that **temporal OEM snapshots** can be obtained by reducing MOEM. We present a system that implements the above, and we follow an example scenario that demonstrates how an underlying MOEM database can accommodate changes in an OEM database. Furthermore, we show that MOEM is capable to model changes occurring not only in OEM databases, but in Multidimensional OEM databases as well. Then, we demonstrate how MQL can be used to pose various queries on the history of evolving OEM and MOEM databases. Finally, we compare our approach with previous work on representing and querying histories of semistructured databases.

6.1 MOEM BASIC CHANGE OPERATIONS

A conventional OEM graph is defined in [Suc98, AQM+97] as a quadruple O = (V, E, r, v), where V is a set of nodes, E a set of labeled directed edges (p, l, q) where $p, q \in$ V and l is a string, r is a special node called the root[97], and v is a function mapping each node to an atomic value of some type (integer, string, etc.), or to the reserved value `C` which denotes a complex object. In order to modify an OEM database O, the following four basic change operations were identified in [CAW99].

- `creNode(nid, val)`: creates a new node, where `nid` is a new node oid (`nid` $\notin$ V), and `val` is an atomic value or the reserved value `C`.

[97] Every node in V must be reachable from the root through some path.

- `updNode(nid, val)`: changes the value of an existing object `nid` to the new value `val`. The node `nid` must not have any outgoing edges (in case its old value is `C`, the edges should have been removed prior to updating the value).
- `addArc(p, l, q)`: adds a new edge labeled `l` from object `p` to object `q`. Both nodes `p` and `q` must already exist in V, and (`p`, `l`, `q`) must not exist in E.
- `remArc(p, l, q)`: removes the existing edge (`p`, `l`, `q`). Both nodes `p` and `q` must exist in V.

Given an MOEM database $M = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$, we introduce the following basic operations for changing M[98]. Remember that $V = V_{mld} \cup V_{cxt}$ is the set of all nodes in M, and $E = E_{cxt} \cup E_{ett}$ is the set of all edges in M.

- `createCNode(cid, val)`: a new context node is created. The identifier `cid` is new and must not occur in V_{cxt}. The value `val` can be an atomic value of some type, or the reserved value `C`.
- `updateCNode(cid, val)`: changes the value of `cid` $\in V_{cxt}$ to `val`. The node must not have any outgoing edges.
- `createMNode(mid)`: a new multidimensional node is created. The identifier `mid` is new and must not occur in V_{mld}.
- `addEEdge(cid, l, id)`: creates a new entity edge with label `l` from node `cid` to node `id`, where `cid` $\in V_{cxt}$ and `id` $\in$ V.
- `remEEdge(cid, l, id)`: removes the entity edge (`cid`, `l`, `id`) from M. The edge (`cid`, `l`, `id`) must exist in E_{ett}.
- `addCEdge(mid, cxt, id)`: creates a new context edge with context `cxt` as label, from node `mid` to node `id`, where `mid` $\in V_{mld}$ and `id` $\in$ V.
- `remCEdge(mid, cxt, id)`: removes the context edge (`mid`, `cxt`, `id`) from M. The context edge (`mid`, `cxt`, `id`) must exist in E_{cxt}.

For OEM and MOEM object deletion is achieved through edge removal, since in both OEM and MOEM the persistence of an object is determined by whether or not the object is reachable from the root.

Sometimes the result of a single basic operation *u* leads to an inconsistent state: for instance, when a new object is created, it is temporarily unreachable from the root. In practice however, it is typical to have a sequence $L = u_1, u_2, ..., u_n$ of basic operations u_i, which corresponds to a higher level modification to the database. By associating such higher level modifications with a timestamp, an OEM history H is defined as a sequence of pairs (*t*, U), where U denotes a set of basic change operations that corresponds [CAW99] to L, and *t* is the associated timestamp. Note that within a single sequence L, a newly created node may be unreachable from the root and still not be considered deleted. At the end of each sequence however, unreachable nodes are considered deleted and cannot be referenced by subsequent operations.

[98] These basic change operations apply not only to MOEMs but also to Multidimensional Data Graphs in general, however in this chapter we focus on MOEMs.

6.2 REPRESENTING OEM HISTORIES WITH MOEM

In this section we give a detailed explanation of the way MOEM can represent changes in an OEM database. Next, we discuss how Multidimensional Data Graph properties, like inherited coverage and reduction, apply in the case of representing OEM histories. In particular, we show that we can use reduction to OEM and partial reduction to obtain temporal snapshots of the OEM database.

6.2.1 Using MOEM to Model OEM Histories

Our approach is to map the four OEM basic change operations to MOEM basic operations, in such a way, that new facets of an object are created whenever changes occur in that object. In this manner, the initial OEM database O is transformed into an MOEM graph[99], that uses a dimension `d` whose domain is time to represent an OEM history H valid [CAW99] for O. We assume that our time domain T is linear and discrete. We also assume:

- A reserved value `start`, such that `start` $<$ `t` for every `t` $\in$ T, representing the beginning of time.
- A reserved value `now`, such that `t` $<$ `now` for every `t` $\in$ T, representing the current time.

The time period during which a context node is the holding node of the corresponding multidimensional entity is denoted by qualifying that context node with a context specifier of the form `[d in {`t_1`..`t_n`}]`. Remember that in context specifiers the syntactic shorthand $v_1..v_n$ for discrete and totally ordered domains means all values v_i such that $v_1 \leq v_i \leq v_n$.

[99] The initial OEM database is transformed to a Multidimensional Data Graph, which, as we will show in Section 6.2.2, is always an MOEM.

Figure 6.1: Modeling OEM basic change operations with MOEM.

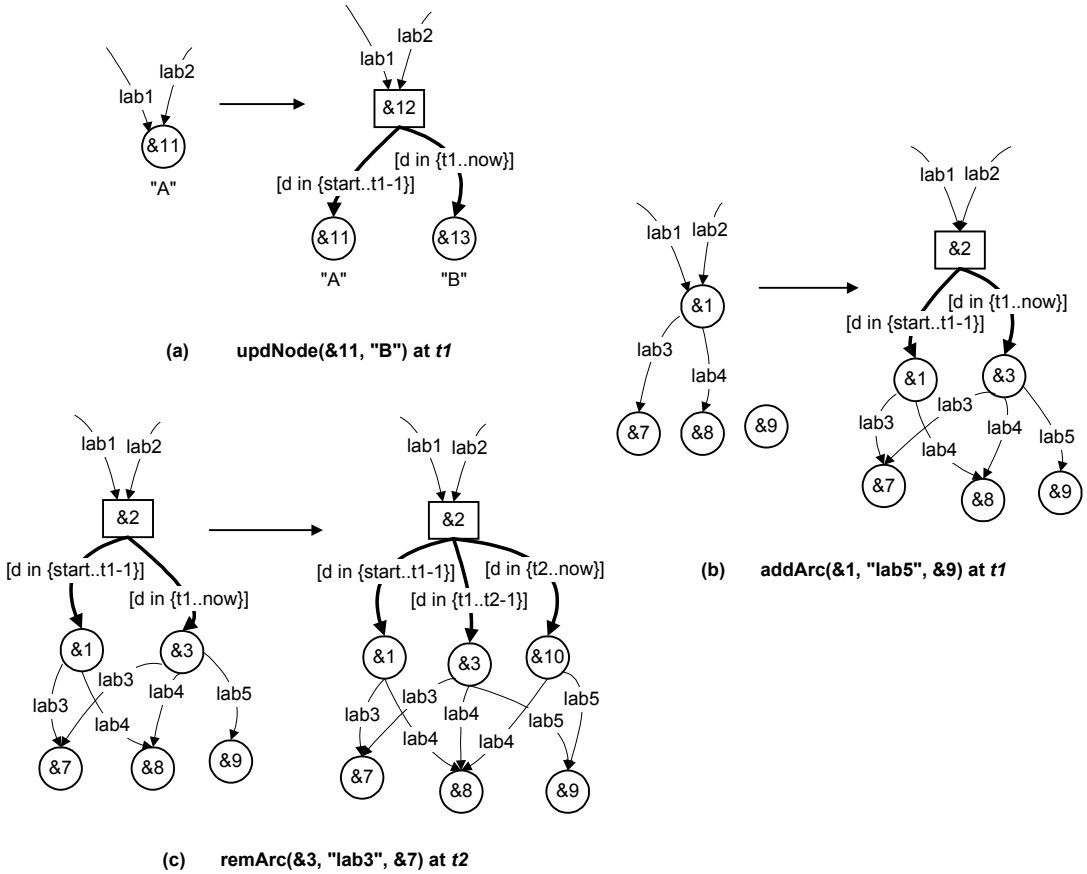


Figure 6.1 gives an intuition about the correspondence between OEM and MOEM operations. Consider the sets U_1 and U_2 of basic change operations, with timestamps t_1 and t_2 respectively. Figure 6.1 (a) shows the MOEM representation of an atomic object, whose value "A" is changed to "B" through a call to the basic change operation `updNode` of U_1. Figure 6.1 (b) shows the result of `addArc` operation of U_1, while Figure 6.1 (c) shows the result of `remArc` operation of U_2, on the same multidimensional entity. It is interesting to notice that three of the four OEM basic change operations are similar, in that they update an object be it atomic (`updNode`) or complex (`addArc`, `remArc`), and all three are mapped to MOEM operations that actually update a new facet of the original object. Creating a new node with `creNode` does not result in any additional MOEM operations; the new node will subsequently be linked with the rest of the graph (within the same set U_i) through `addArc` operation(s), which will cause new object facet(s) to be created. It is worth noting that the changes induced by the OEM basic change operations affect only localized parts of the MOEM graph, and do not propagate throughout the graph.

As shown in Figure 6.1, an OEM object may correspond to a number of MOEM objects with different oids, which is perceived as an oid change through the history of the OEM object. However, this is more an implementation issue and does not present any real problem.

Having outlined the approach, we now give a detailed specification. First, the following four utility functions and procedures are defined on MOEM.

1. `id1` ← `md(id2)`, with `id1`, `id2` ∈ V. Returns the multidimensional node for a context node, if it exists. If `id2` ∈ V_{cxt} and there exists an element (`mid`, `cxt`, `id`) in E_{cxt} such that `id = id2`, then `mid` is returned. If `id2` ∈ V_{cxt} and no corresponding context edge exists, `id2` is returned. If `id2` ∈ V_{mld}, `id2` is returned. We assume that there is at most one multidimensional node pointing to any context node, in other words for every `cid` ∈ V_{cxt} there is at most one `mid` such that (`mid`, `cxt`, `cid`) ∈ E_{cxt}. Notice, however, that this is a property of MOEM for the specific problem because of the special way the MOEM graph is constructed for representing histories, and does not apply to the general case.

2. `boolean` ← `withinSet(cid)`, with `cid` ∈ V_{cxt}. Checks whether the context node `cid` is created within the current set U of basic change operations. This function is used while change operations are in progress, and returns `true` if `cid` was created within the same set. It returns `false` if `cid` was created within a previous set of operations.

3. The following procedure `mEntity(id)`, with `id` ∈ V_{cxt}, creates a new multidimensional node `mid` pointing to `id`, and redirects all incoming edges from `id` to `mid`. Note that the procedure alters the graph, but not the information modeled by the graph: the multidimensional entity created by the procedure has node `id` as its only facet holding under every world[100].

```
mEntity(id) {
    createMNode(mid)
    addCEdge(mid, [d in {start..now}], id)
    for every (x, l, id) in Eett {
        addEEdge(x, l, mid)
        remEEdge(x, l, id)
    }
}
```

4. In the procedure `newCxt(id1, id2, ts)`, with `id1`, `id2` ∈ V_{cxt} and `ts` ∈ T, `id1` is the currently most recent facet of a multidimensional entity, and `id2` is a new facet that is to become the most recent. The procedure arranges the context specifiers accordingly.

```
newCxt(id1, id2, ts) {
    remCEdge(md(id1), [d in {x..now}], id1)
    addCEdge(md(id1), [d in {x..ts-1}], id1)
    addCEdge(md(id1), [d in {ts..now}], id2)
}
```

The next step is to show how each OEM basic change operation is implemented using the basic MOEM operations. We assume that each of the OEM operations is part of a set U with timestamp `ts`, and that the node `p` is the most recent context node of the corresponding multidimensional entity, if such an entity exists. This is because changes always happen to the current snapshot of OEM, which corresponds to the most recent facets of MOEM multidimensional entities. The most recent context node is the one holding in current time, i.e. the node whose context specifier is of the form `[d in {`*somevalue*`..now}]`.

[100] The initial graph and the resulting graph have the same canonical form.

- `updNode(p, newval)` OEM change operation:

If `p` has been created within U, its value is updated directly, and the process terminates. Otherwise, if `p` is not pointed to by a multidimensional node, a new multidimensional node is created for `p`, having `p` as its only context node with context specifier `[d in {start..now}]`. A new facet is then created with value `newval`, and becomes the most recent facet by adjusting the relevant context specifiers. Since a node updated by `updNode` cannot have outgoing edges, no edge copying takes place in contrast to the case of `addArc`.

```
updNode(p, newval) {
    if not withinSet(p) {
        if not exists (x, cxt, p) in E_cxt
            mEntity(p)
        createCNode(n, newval)
        newCxt(p, n, ts)
    }
    else updateCNode(p, newval)
}
```

- `addArc(p, l, q)` OEM change operation:

If `p` has been created within U, it is used directly: the new edge is added, and the process terminates. Otherwise, if `p` is not already pointed to by a multidimensional node, a new multidimensional node is created for `p`, having `p` as its only context node with context specifier `[d in {start..now}]`. A new "clone" facet `n` is then created, by copying all outgoing edges of `p` to `n`. In this case, the context specifiers are adjusted so that `ts` is taken into account, and `n` becomes the most recent facet as depicted in Figure 6.1 (b) for `ts = t1`. Finally the new edge specified by the basic change operation is added to the most recent facet. Note that, in the frame of representing changes, an MOEM is constructed in such a way that an entity edge does not point directly to a context node q_c if there exists a context edge (q_m, `cxt`, q_c); instead, it always points to the corresponding multidimensional node q_m, if q_m exists. This is achieved by using the function `md(q)` in combination with `mEntity(p)`.

```
addArc(p, l, q) {
    if not withinSet(p) {
        if not exists (x, cxt, p) in E_cxt
            mEntity(p)
        createCNode(n, 'C')
        newCxt(p, n, ts)
        for every (p, k, y) in E_ett
            addEEdge(n, k, y)
        addEEdge(n, l, md(q))
    }
    else addEEdge(p, l, md(q))
}
```

- `remArc(p, l, q)` OEM change operation:

The process is essentially the same as `addArc(p, l, q)`, with the difference of removing an edge at the end of the process, instead of adding one. Therefore, `remArc` is like `addArc` except for the last two calls to `addEEdge`, which are replaced with calls to `remEEdge` with the same arguments.

- `creNode(p, val)` OEM change operation:

  This basic change operation is mapped to `createCNode(p, val)` with no further steps. New facets will be created when new edges are added to connect node `p` to the rest of the graph.

6.2.2 Applying MSSD Properties

In the previous section we explained how the history of an OEM database is incorporated in an MOEM graph, using multidimensional nodes and context edges. However, we have not shown that this graph is actually an MOEM graph. It is obviously a Multidimensional Data Graph, but in order to be an MOEM graph (a) it must be context-deterministic, and (b) every node and edge must have a non-empty inherited coverage.

In this section we examine how the MSSD properties of context-determinism, inherited coverage, and reduction are applied in the particular case of representing OEM histories. We show that a Multidimensional Data Graph constructed as specified in Section 6.2.1 is always an MOEM, and furthermore, an MOEM that has special characteristics not generally encountered in MOEMs.

Let G be a Multidimensional Data Graph produced by the process specified in Section 6.2.1, let *e* be a multidimensional entity in G, with multidimensional node *m* and facets e_1, e_2, ..., e_n, and let c_1, c_2, ..., c_n be the context specifiers of the respective context edges. Then, the process in Section 6.2.1 guarantees the following *special properties* for G:

(a) Context edges always point to context nodes, thus e_1, e_2, ..., e_n are exclusively context nodes.

(b) A context node e_i is facet of at most one multidimensional node *m*, therefore at most one context edge can point to e_i.

(c) Every multidimensional node *m* (except possibly the root) is pointed by at least one entity edge.

(d) A time instance is a world for G.

(e) Leaf nodes in G are exclusively atomic nodes[101].

Because of the procedure `newCxt`, which constructs the context edges, for every multidimensional entity *e* in G the explicit contexts c_1, c_2, ..., c_n always define disjoint sets of worlds. Consequently, the inherited coverages of the context edges are mutually exclusive, since they are context subsets of the corresponding explicit contexts. Because of the special property (a) above, given any world w, if we start from *m* we can navigate to at most one context node through context edges that hold under w, hence G is context-deterministic.

In addition, from the procedures `mEntity` and `newCxt` it can be seen that:

(f) c_1 has the form `[d in {start..`*somevalue_1*`}]`

(g) c_n has the form `[d in {`*somevalue_N*`..now}]`

(h) The context union of c_1, c_2, ..., c_n is `[d in {start..now}]`, for every *e* in G.

Although for every *e* in G the explicit contexts c_1, c_2, ..., c_n cover the complete `{start..now}` time range, this is not the case with the corresponding inherited coverages,

[101] This holds assuming that all successive states in a database history are non-empty OEMs whose leaves are atomic nodes.

which denote the true life span of the entity and its facets. First, lets discuss context coverage. Because of the special properties (c), (e), and (h) above, the context coverage of all context nodes, multidimensional nodes, and entity edges in G is `[d in {start..now}]`, which is a universal context. Besides, the context coverage of a context edge is always its explicit context. Therefore, in the case of representing OEM histories context coverage does not propagate any constraints, and for every node and edge in G *the inherited coverage coincides with the inherited context*.

Lets examine the meaning of inherited context in G. Each multidimensional entity *e* in G corresponds to a node that existed at some time in the evolution of the OEM graph. The facets of *e* correspond to OEM changes that had affected that node. Edges pointing to *m* correspond to edges that pointed to that node at some time in the evolution of the OEM graph. In addition, the inherited context of edges pointing to *m* will be such as to allow to each one of e_1, e_2, ..., e_n to "survive" under some world. Therefore, for every e_i with $2 \le i \le n-1$ the explicit context c_i is also the inherited context of the context node e_i. As we have seen, c_1 = `[d in {start..`*somevalue_1*`}]`, and c_n = `[d in {`*somevalue_N*`..now}]`; for facets e_1 and e_n incoming edges restrict the explicit contexts, so that the inherited context of e_1 may have a first value greater than `start`, while the inherited context of e_n may have a second value smaller than `now`.

It is now easy to see that there cannot be any elements in G with empty inherited coverage, and, since G is context-deterministic, G is an MOEM. In addition, the inherited coverage of the root of G is a universal context, meaning that for any time instance `t` within `{start..now}` we can reduce G to an OEM holding under `t`.

Consequently, given an OEM database O and an MOEM database M that represents the history of O, it is possible to specify any time instance `t` from the time domain T and *reduce* M *to an OEM database* O′ holding under `t`. Then O′ will be the snapshot of O at the time instance `t`. It is also possible to *apply partial reduction* to M for a set of time instances `[d in {`t_1`,`t_2`,...,`t_n`}]`. In this case an MOEM sub-graph of M is returned, encompassing the OEM snapshots at $t_1, t_2, \ldots, t_n$.

6.3 IMPLEMENTATION: "OEM HISTORY"

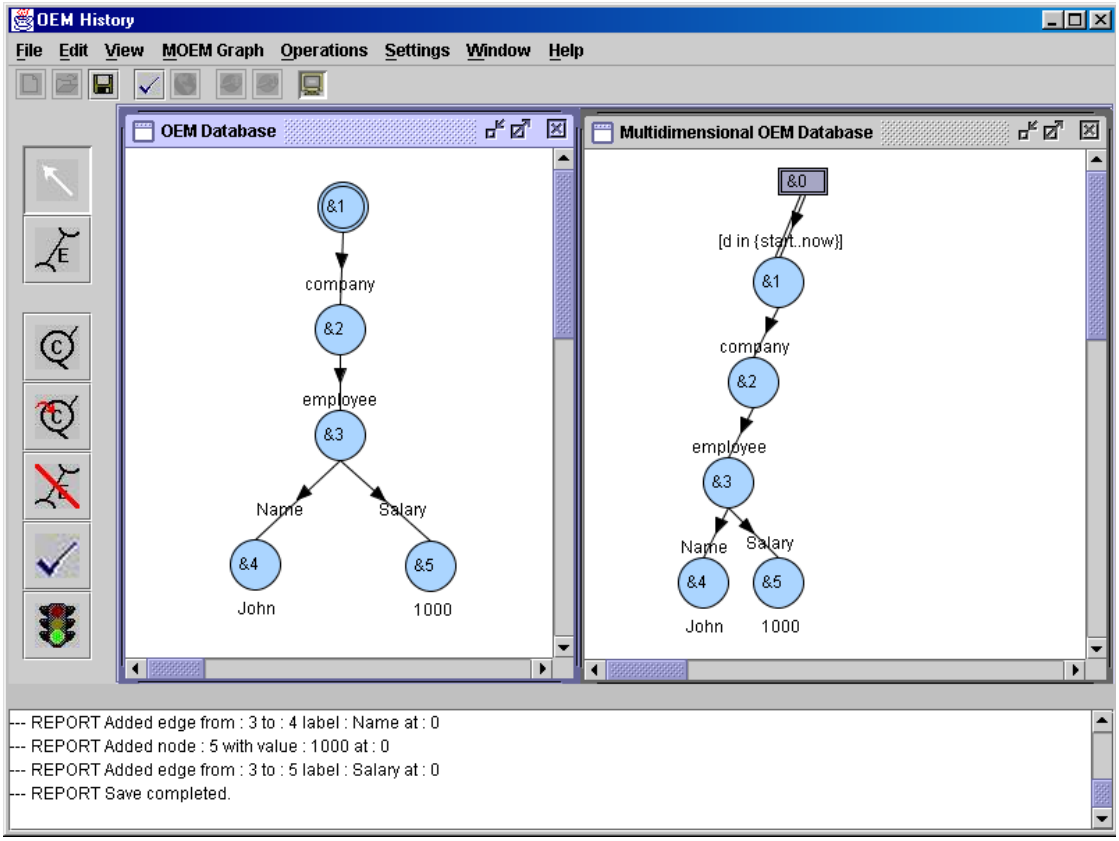
OEM History is an application developed in Java [JAVA, CL97], which implements the method described in Section 6.2 for representing OEM histories. As it can be seen in Figure 6.2, OEM History employs a multi-document interface (MDI) with each internal window displaying a data graph. There are two main windows: one that displays an MOEM graph that corresponds to the internal model of the application, and one that always shows the current state of the OEM database. Furthermore, the user can ask for a snapshot of the database for any time instance in T (the time domain), which will be presented as an OEM graph in a separate window. The toolbar on the left side contains buttons that correspond to the four OEM basic change operations, and can be used only on the window with the OEM depicting the current state of the database. In reality, these buttons invoke operations that update the internal MOEM data model[102] of the application, which is the only model actually maintained by OEM History. The current OEM database is the result of an MOEM *reduction to OEM* under the world where `d` is `now`.

The "tick" button in the left toolbar removes nodes that are not accessible from the root. The last button in the toolbar marks the end of a sequence of basic change operations, and

[102] As specified in Section 6.2.1.

commits all changes to the database under a common timestamp. Operations like MOEM reduction to OEM can be initiated from the upper toolbar or from the application menu.

Figure 6.2: Initial state of example database in *OEM History* application.



In Figure 6.2 we see the initial state of an OEM database containing information about the employees of a company, and the corresponding MOEM graph. The right window displays the underlying MOEM model, while the left window displays the result of an MOEM reduction for `d=now`.

Figure 6.3: Example database after a couple of sequences of basic changes upon the initial database state.

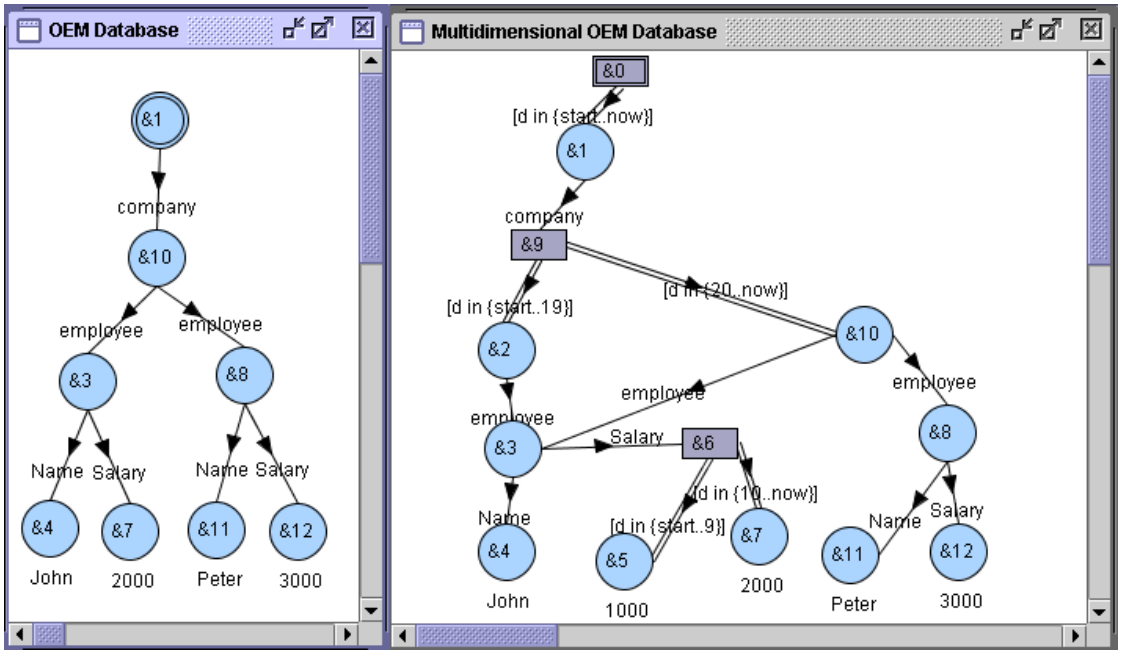
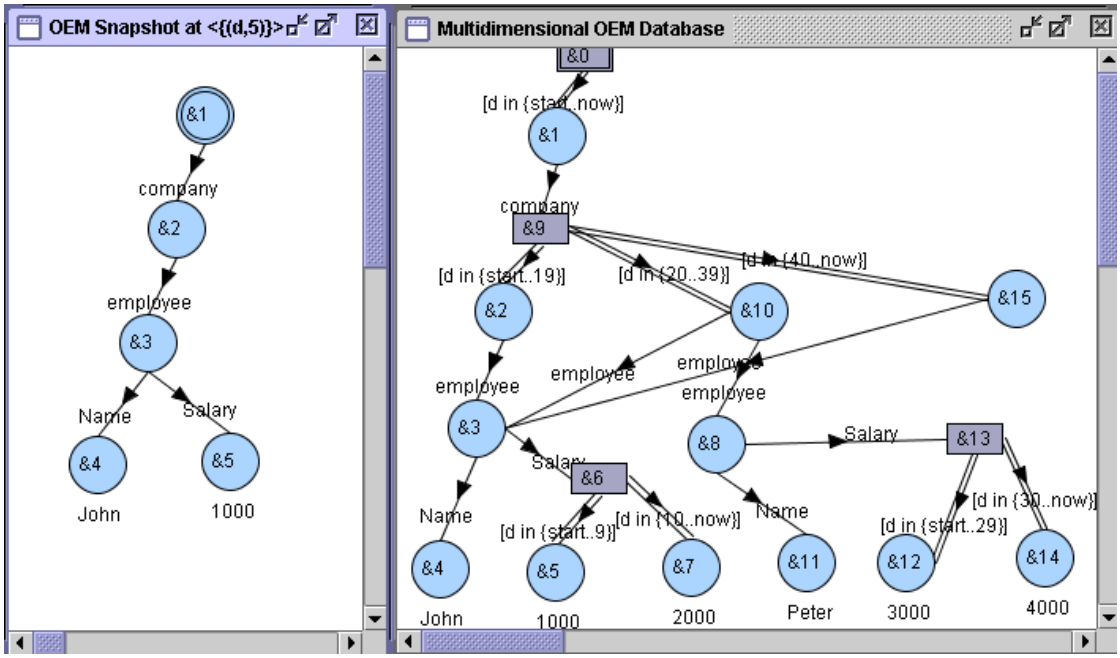


Figure 6.3 shows the current state of the OEM database and the corresponding MOEM graph after a couple of change sequences. First, at the time instance `10` the salary of `John` has been increased from `1000` to `2000`. Then, at the time instance `20` a new employee called `Peter` joined the company with salary `3000`.

Figure 6.4: Example database after two more sequences of basic changes upon the database of Figure 6.3.



In Figure 6.4 two more change sequences have been applied. The salary of `Peter` increased to `4000` at the time instance `30`, but quite ungratefully `Peter` left the company at the time instance `40`. Note that, as shown on the caption, the left window does not display the current OEM. Instead it depicts a snapshot of the OEM database for the time instance `5`, which is obtained by reducing to OEM the MOEM in the right window for `d=5`. That snapshot is identical to the initial state of the database, since the first change occurred at the time instance `10`.

6.4 REPRESENTING CHANGES IN MOEM DATABASES

Besides representing the history of OEM databases, MOEM has another interesting property. In this section we show that MOEM is expressive enough to *model its own histories*. In other words, for any MOEM database M evolving over time it is possible to maintain an MOEM database M′, such that M′ represents the history of M[103].

The approach is similar to that of Section 6.2.1; we show that each of the MOEM basic change operations that are applied to M, can be mapped to a number of MOEM basic change operations on M′, in such a way that M′ represents the history of M. Figure 6.5 gives the intuition about this mapping, for three basic operations. Context edge labels `c1`, `c2`, ..., `cN` denote context specifiers involving any number of dimensions, while the dimension `d` is defined in Section 6.2.1. Note that the use of dimension `d` in M′ does not preclude M from using other dimensions that range over time domains. The MOEM operations depicted in Figure 6.5 are basic operations occurring on M, and the corresponding graphs show how those operations transform M′. For simplicity, graphs on the left side do not contain context specifiers with the dimension `d`, and all timestamps are `t1`. It is however easy to envisage the case where `d` is also on the left side and timestamps progressively increase in value, if we look at Figure 6.1 (b) and Figure 6.1 (c) which follow a similar pattern.

[103] In general, a Multidimensional Data Graph G′ can be constructed so as to represent the changes occurring on another Multidimensional Data Graph G. Like in Section 6.2.2 for OEM, it can be shown that G′ is an MOEM assuming that all successive states in the history of G are non-empty MOEMs.

Figure 6.5: Modeling MOEM basic change operations with MOEM.

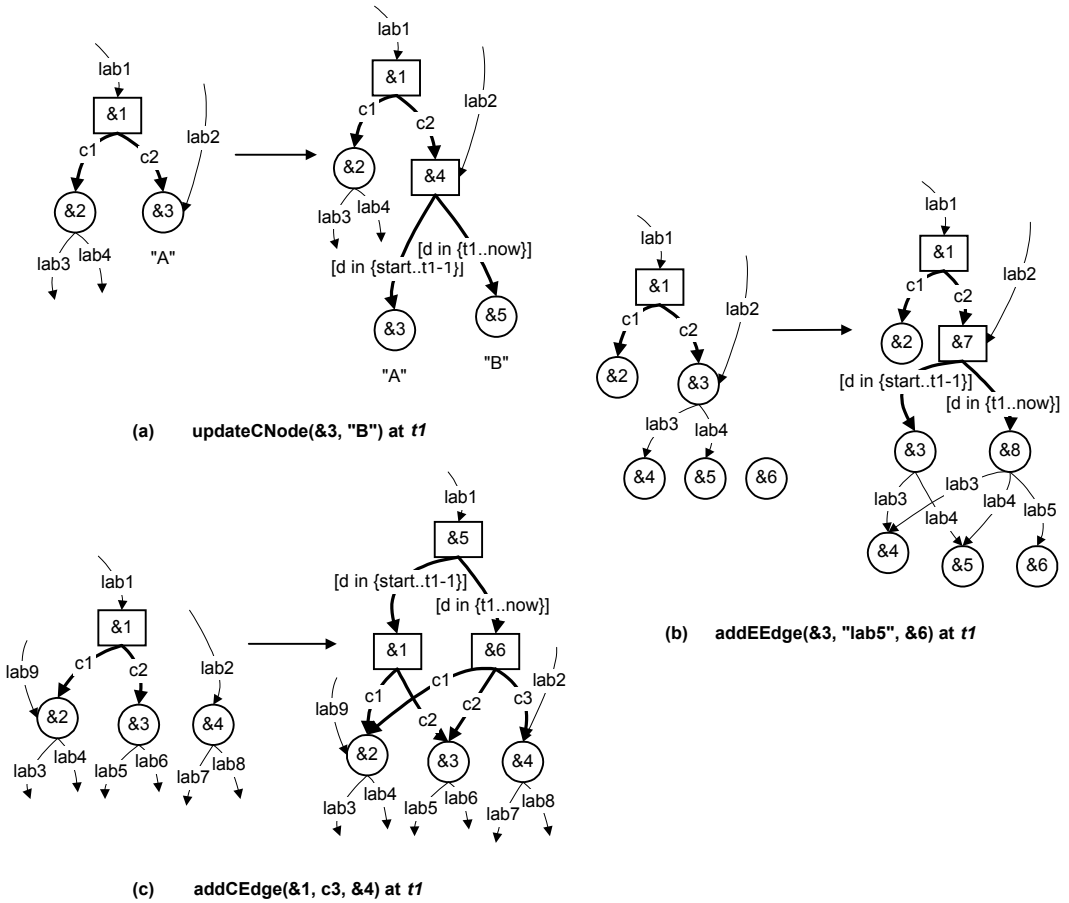


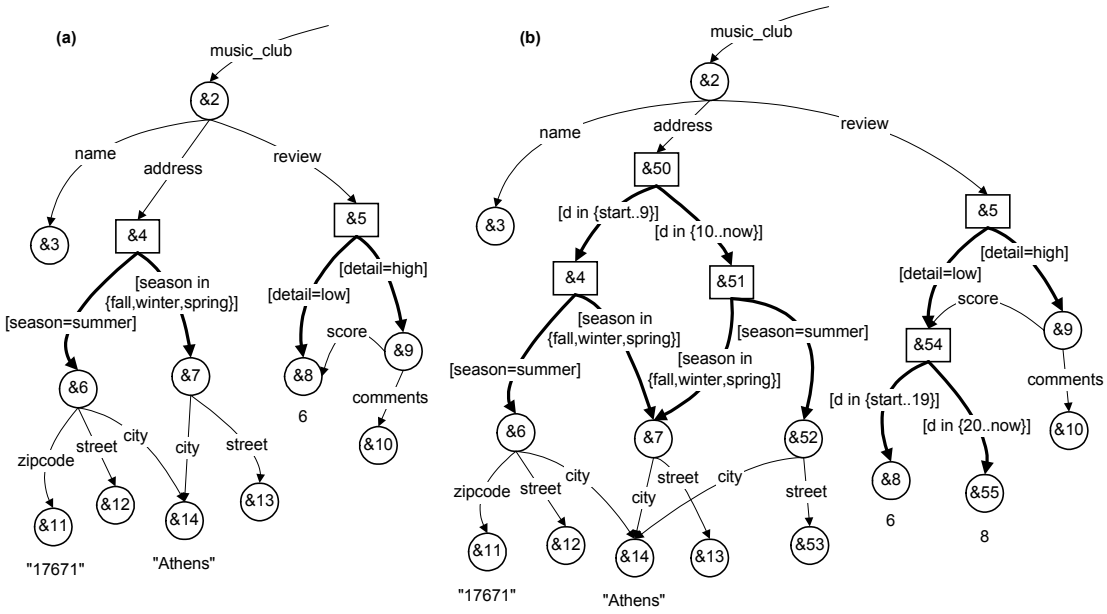
Figure 6.5 (a) shows a facet with oid &3 whose value is changed from `"A"` to `"B"` through a call to `updateCNode`. Figure 6.5 (b) shows the result of an `addEEdge` operation. Finally, Figure 6.5 (c) depicts the `addCEdge` basic operation. Among MOEM basic operations *not* shown in Figure 6.5, `remEEdge` is very similar to `addEEdge`; the difference is that an entity edge is removed from facet &8 instead of being added. In addition, `remCEdge` is similar to `addCEdge`: instead of adding one context edge to &6, one is removed. Finally, the MOEM basic operations `createCNode` and `createMNode` are mapped to themselves; M′ will record the change when the new nodes are connected to the rest of the graph M, through calls to `addEEdge` or `addCEdge`.

An MOEM graph M′ constructed through the process outlined above represents the history of the MOEM database M. In contrast to the case of OEM histories where a world is defined by giving a value to a single dimension `d` representing time, in the case of MOEM histories a world for M′ involves in general more than one dimensions, including the time dimension `d`. Therefore, by specifying a time instance `t` we actually define the set of worlds for which `d=t`. In that set, dimensions other than `d` may have any combination of values from their respective domains.

In order to get a temporal snapshot of M from M′, *partial reduction is used*. By applying the process of partial reduction to M′ for any time instance $t \in T$, we get the snapshot of the

MOEM database M at t[104]. It is also possible to apply partial reduction to M′ for a set of time instances `[d in {`t_1`,`t_2`,...,`t_n`}]`. In this case an MOEM sub-graph of M′ is returned, encompassing all MOEM snapshots at $t_1, t_2, \ldots, t_n$.

Figure 6.6: Representing the history of the multidimensional music club in (a) with the MOEM in (b).



As an example, consider the MOEM graph of Figure 6.6 (b), which represents the history of the multidimensional music club in Figure 6.6 (a). The initial state of the database as depicted in Figure 6.6 (a) does not contain the nodes &50, &51, &52, &53, &54, and &55 as well as their outgoing edges, which are introduced in Figure 6.6 (b) because of change operations. Two sequences of changes have occurred: (1) at the time instance `10` the summer address of the club changed to another address with a different `street` and without a `zipcode`, and (2) at the time instance `20` the review score changed from `6` to `8`. The first sequence of changes involves the redirection of a context edge, so that it points to the newly added node &52 instead of node &6. The redirection is achieved through a call to `remCEdge` and a subsequent call to `addCEdge`, which are addressed in Figure 6.5 (c). The second sequence of changes is actually a call to `updateCNode` for the node &8, which is addressed in Figure 6.5 (a).

6.5 QUERYING CHANGES WITH MQL

The approach we presented above keeps different "versions" of each object as they change, encompassing their histories within the corresponding multidimensional entities as a

[104] To be strict, another step must be taken to obtain a temporal snapshot of the database: context edges with explicit context concerning the dimension `d` are not significant any more and must be removed. This step can be considered as similar to **de-contextualization**, defined in Chapter 4.

collection of object snapshots. It is interesting to notice that a simpler representation could also have allowed for reduction of the database to temporal snapshots: context edges that depart from a multidimensional root lead to the graphs of all database states that have occurred, and they are labeled with the valid period of the corresponding database state. In effect, this method keeps a copy of the whole database after each sequence of basic change operations, and is clearly a waste of space, but it allows reduction in a straightforward way. An important benefit of representing changes at the level of multidimensional entities rather than as a collection of database snapshots stems from the queries we can formulate to retrieve information about the evolution of those entities.

In this section we give examples of such queries, and demonstrate how MQL can be used in the frame of the current problem. We start with querying histories of OEM databases, and continue with querying histories of MOEM databases.

6.5.1 Querying OEM Histories

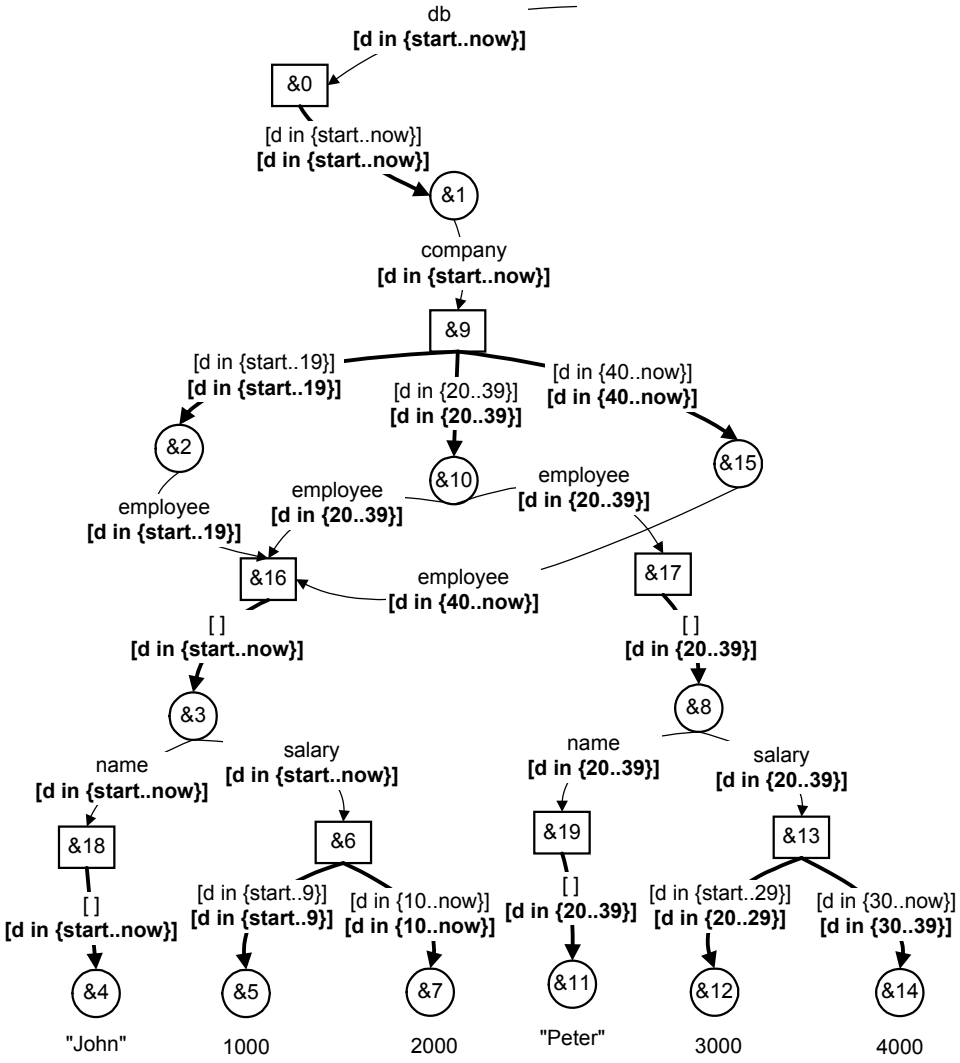
For our query examples, we will use the MOEM database depicted in Figure 6.4. Context path expressions in MQL queries are matched against the canonical form of an MOEM database, and they often use path inherited coverage, which is a powerful property of Multidimensional Data Graphs. To facilitate the comprehension of queries, Figure 6.7 depicts the database of Figure 6.4 transformed in canonical form and supplemented with the inherited coverages of edges, which appear in boldface characters. As shown in Figure 6.7, the transformation to canonical form introduced the multidimensional nodes &16, &17, &18, and &19.

It is important to note that the transformation of the underlying MOEM database to canonical form can take place at any stage during the evolution of an OEM database. The process specified in Section 6.2.1 for modeling OEM histories can be equally applied whether or not the underlying MOEM is in canonical form, with a single exception: if the MOEM is in canonical form, we need to maintain this canonical form when new OEM nodes are created. To cover this exception, a simple extension must be incorporated: whenever a new context node is created as a result of `creNode(p,val)`, a multidimensional node and a context edge pointing to that context node are created as well.

In effect, if the MOEM database is in canonical form, then the process of Section 6.2.1 is simplified considerably. Specifically, it will no longer be necessary to call the procedure `mEntity(id)` to create a multidimensional node, because the transformation to canonical form will have introduced all the multidimensional nodes that are needed at one go[105].

[105] A minor point is that context edges created by transformation to canonical form have explicit context the universal context `[]`, while context edges created by `mEntity(id)` have explicit context `[d in {start..now}]`, which in our case is also a universal context representing every possible world.

Figure 6.7: The MOEM database of Figure 6.4 in canonical form, supplemented with the inherited coverages of edges.



Multidimensional entities in the database of Figure 6.7 model OEM objects that may have been submitted to changes, and MQL queries on those entities can be interpreted as queries on the history of the corresponding OEM objects.

- It is simple to get the value of an OEM object at a specific time instance.

The query that follows returns the salary of Peter at time instance 32:

```
select salary: S
from [d=32]db.company.employee{X}.salary S
where X.name = "Peter"
```

The result of the query is the mssd-expression `{salary: &14 4000}`. The context path expression `[d=32]db.company.employee.salary` is equivalent to the conventional path expression `db.company.employee.salary` evaluated on the conventional OEM that holds under time instance 32. Therefore, the object variable `S` binds to the `salary` facet that exists

in that OEM. The same result would have been returned if, for example, we had used `[d in {32..36}]` instead of `[d=32]`, because the path leading to node &14 holds under all time instances from 32 to 36. However, if we had used `[d in {25..50}]`, no `salary` object would have been returned, because no single `salary` facet holds under the time period from 25 to 50.

A similar query returns the name(s) of the employee(s) whose salary at time instance 25 was greater than or equal to 2500:

```
select name: Y
from [d=25]db.company.employee{X}.salary S,
     X.name Y
where S >= 2500
```

The result is `{name: &11 "Peter"}`.

- We can retrieve information concerning the history of an OEM object.

The query below returns Peter's salaries, together with the periods they were valid:

```
select [Y]: S
from db.company.employee{X}.salary::[Y][-] S
where X.name = "Peter"
```

The context variable `[Y]` binds to the inherited coverage of context edges leading to `salary` facets, and is used to turn this into explicit context in the results:

```
<[d in {20..29}]: &12 3000,
 [d in {30..39}]: &14 4000>
```

The result graph consists of a multidimensional root from which a couple of context edges depart, leading to the context nodes &12 and &14. Observe that the inherited coverage of the context edge leading to node &12 in the database is `[d in {20..29}]`, and reflects the fact that Peter joined the company at the time instance 20. In a similar way, the inherited coverage of the context edge leading to node &14 in the database reflects the fact that Peter left the company at the time instance 40. Both inherited coverages appear as explicit contexts in the results.

The following query retrieves the time instances at which some change occurred in Peter's salary:

```
select period_without_change: [Y]
from db.company.employee{X}.salary::[Y][-]
where X.name = "Peter"
```

The result is:

```
{period_without_change: "[d in {20..29}]",
 period_without_change: "[d in {30..39}]"}
```

The result indicates that at time instance 30 a change occurred in the salary of Peter. Notice how the context variable `[Y]` becomes the string value of new atomic objects in the result.

The smallest and greatest values (20 and 39) of `d` in the above result denote the life span of the `salary` multidimensional entity, which is this case happens to be a continuous interval. It is possible to get the life span of a multidimensional entity directly, as shown by the query that follows:

```
select lifespan: [Z]
context [Z] as union([Y])
from db.company.employee{X}.salary::[Y][-]
where X.name = "Peter"
```

The clause `context` is used to introduce a new context variable `[Z]` as the context union of all values bound to `[Y]`. The result is: `{lifespan: "[d in {20..39}]"}`.

- We can get the values of an OEM object that were valid anytime within a period.

The following query looks for Peter's salaries in the period between 25 and 50.

```
select [Y]: S
from [Y]db.company.employee{X}.salary S
where X.name = "Peter"
within [Y] * [d in {25..50}] != [-]
```

For every value of `S`, the context variable `[Y]` binds to a path inherited coverage, which represents the time instances under which OEMs hold, containing a path leading to this value of `S`. Then, `[Y]` is used in the `within` clause to filter out `salary` facets that do not hold under any of the time instances between 25 and 50. The query returns:

```
<[d in {20..29}]: &12 3000,
 [d in {30..39}]: &14 4000>
```

Had we used a time instance greater than 29 instead of the time instance 25, only node &14 would have been returned. For the period between 40 and 50 no salary exists, because at time instance 40 Peter left the company. Notice the difference from a previous example that looks for a salary holding under the whole period from 25 to 50.

- We can find out when an OEM object had a specific value.

The query below returns the period during which John's salary was 2000.

```
select valid_time: [Y]
from [Y]db.company.employee{X}.salary S
where X.name = "John" and S = 2000
```

The result is:

```
{valid_time: "[d in {10..19}]",
 valid_time: "[d in {20..39}]",
 valid_time: "[d in {40..now}]"}
```

The context variable `[Y]` binds to *path* inherited coverages, and since there are three paths in the database leading to node &7, there exist three objects in the result. Taken together, they give the period during which the salary of John was 2000. If we want the whole period as a single object, we can use a `context` clause introducing a new variable as `union([Y])`. Alternatively, we can use a different context path expression in the `from` clause:

```
select valid_time: [Y]
from db.company.employee{X}.salary::[Y][-] S
where X.name = "John" and S = 2000
```

In this case, `[Y]` corresponds to a path that consists of one context edge leading to a `salary` facet, and the result will be `{valid_time: "[d in {10..now}]"}`, which again gives the period during which the salary of John was 2000.

- It is possible to pose general questions on the history of the OEM database.

The query that follows finds all OEM objects that have not changed since their creation:

```
select constant_object: {value: Z, period: [Y]}
from db.#.- <X>
where count(<X>::[Y][-]{Z}) = 1
```

The query uses the wildcard # that matches zero or more *entity edge* / *context edge* pairs, and the wildcard – that matches any single entity edge. The object variable <X> binds to every multidimensional object in the database, except for the one that corresponds to the root. Then, the `where` clause filters out those multidimensional objects that have more than one facet. Remember that in an MOEM that is in canonical form, every context object is the facet of some multidimensional object, and every multidimensional object has at least one associated context object facet.

The same result is returned by the query

```
select constant_object: {value: Z, period: [Y]}
from db.#.-::[Y][] Z
```

which exploits the fact that if an OEM object has not been changed, then the corresponding MOEM context object is pointed to by a context edge whose explicit context is a universal context.

As another example, the following query returns the employees whose salary has not changed since time instance 32.

```
select employee_data: {name: Z, salary: S, valid_period: [Y]}
from db.#.employee{X}.salary::[Y][-] S,
     X.name Z
within [Y] >= [d in {32..now}]
```

The result is:

```
{employee_data: {name: &4 "John",
                 salary: &7 2000,
                 valid_period: "[d in {10..now}]"}
}
```

The `within` clause checks whether the period under which a `salary` facet holds is context superset of the period from 32 until the present time. Notice that node &14 is not included in the results, because the inherited coverage correctly indicates the fact that at the time instance 40 the salary ceased to exist.

- We can combine values and valid times of different OEM objects to formulate queries like "find the value(s) of an object while another object was having the specified value(s)".

The following query returns the salary(-ies) of John, at the time Peter's salary was 3000:

```
select john_salary: S1
from db.company X,
     X.employee{Y1}.salary::[Z1][-] S1,
     X.employee{Y2}.salary::[Z2][-] S2
where Y1.name = "John" and Y2.name = "Peter" and S2 = 3000
within [Z1] * [Z2] != [-]
```

The result of the query is `{john_salary: &7 2000}`.

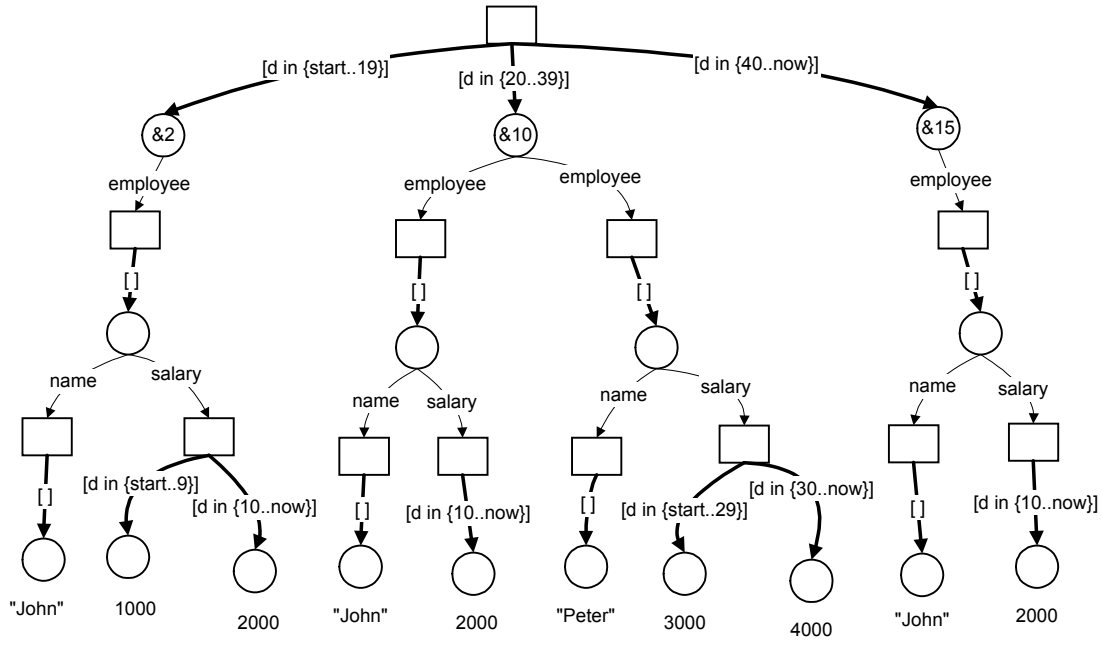
- It is possible to "decompose" a multidimensional entity along the dimension `d`.

The following query gives the snapshots over time of a company called "ACME":

```
select holding <[Y]: Z>
from db.company{<X>}::[Y][-] Z
where <X>.company_name = "ACME"
```

The object variable `<X>` binds to the `company` multidimensional object for which there exists some company facet with the name "ACME"[106]. Variables `[Y]` and `Z` bind to inherited coverages and company facets respectively, and are used in `select` to construct a new multidimensional entity, in which inherited coverages become explicit contexts[107]. The keyword `holding` causes nodes and edges that have empty inherited coverage in the result graph to be removed. In effect, `holding` causes a partial reduction of the subgraphs bound to `Z` for the corresponding contexts bound to `[Y]`. Assuming that the name of the company in the database of Figure 6.7 is "ACME", the final result is depicted in Figure 6.8[108].

Figure 6.8: Partial reductions for various contexts as MQL results.



The graph in Figure 6.8 contains three subgraphs, one for each state in the evolution of the company. The first subgraph has node &2 as a root, and corresponds to the state of the

[106] In this way, company facets that do not point to an "ACME" `company_name` will also be included in the results, as long as there exists a facet pointing to "ACME" in the same multidimensional entity. Compare this to the case the `where` clause contained: `Z.company_name = "ACME"`.

[107] In the database of Figure 6.7, the inherited coverages of edges leading to company facets happen to be the same as the explicit contexts.

[108] As mentioned in the previous chapter, to facilitate the comprehension of queries we indicate in query results the oids of objects as they appear in the database. Keep in mind however, that oids in query results are not necessarily the same as oids of the corresponding objects in the database. This is clearly shown in Figure 6.8, where the same database object is repeated more than once in the results, and therefore cannot have the same oid.

company from the beginning until time instance 19. At that time, the company had only one employee called “John”. The second subgraph has node &10 as a root, and corresponds to the period from 20 to 39. During that period, the company had two employees, “John” and “Peter”. Notice that “John” contains only one salary now, since his first salary is not relevant to this period. The third subgraph has node &15 as a root, and corresponds to the period from 40 until the present time. During this period, “John” is the only employee of the company with a salary that remains unchanged.

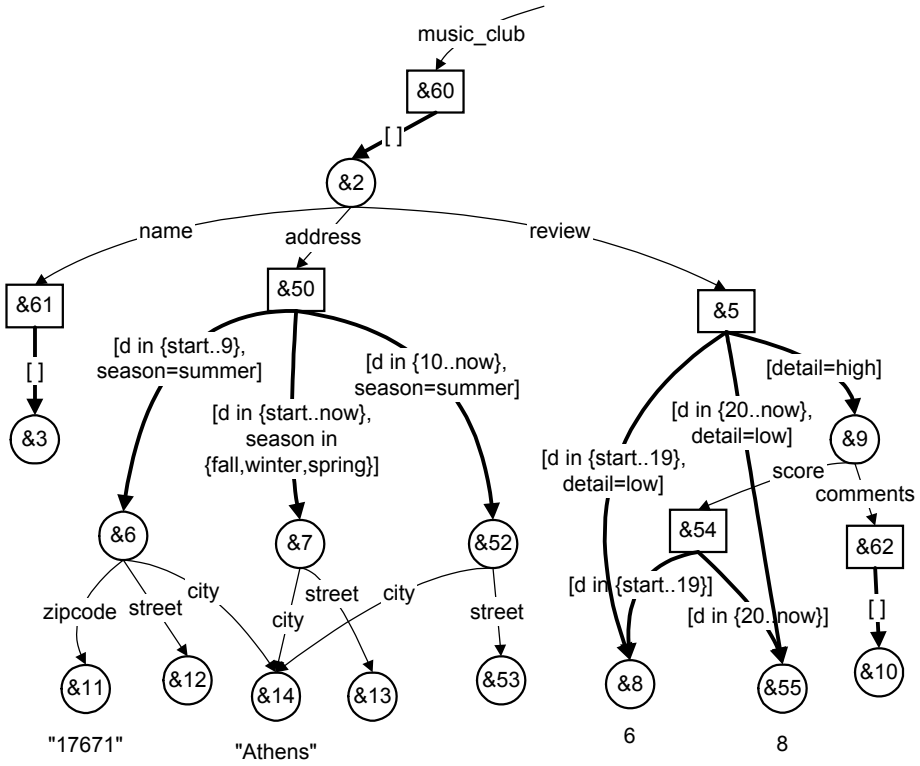
Observe that if the reduction context becomes a time instance, then every multidimensional node in the corresponding subgraph will point to exactly one context node, and partial reduction will be analogous to reduction to OEM under that time instance.

6.5.2 Querying MOEM Histories

In addition to querying OEM histories, MQL can be used to query histories of MOEM databases. In this section we present some example queries on MOEM histories, focusing on the differences from querying OEM histories.

For our example queries we will use the MOEM depicted in Figure 6.9, which in fact is the MOEM of Figure 6.6 (b) transformed to canonical form. Observe that the transformation to canonical form has introduced three new multidimensional nodes, namely &60, &61 and &62, and has removed the multidimensional nodes &4 and &51.

Figure 6.9: The MOEM of Figure 6.6 (b) in canonical form.



The following "plain vanilla" query returns the review of the music club at time instance 15 in low detail:

```
select review: X
from [d=15,detail=low]music_club.review X
```

The result is the mssd-expression {review: &8 6}. This simple query specifies values for all dimensions related to review, and gets a review facet as a result. However, we may be interested in specifying constraints for some dimensions only. As we can see in Figure 6.9, when representing MOEM histories the dimension d is normally used together with other dimensions in context specifiers. We often want to "isolate" this dimension in queries and pose constraints to that dimension alone, irrespective of values of other dimensions. An elegant way to achieve that is to use **context patterns**, which were introduced in the previous chapter. The query below retrieves the music club address(es) at time instance 15, irrespective of the values of other dimensions:

```
select address: X
from [% d=15]music_club.address X
```

The context pattern [% d=15] matches path inherited coverages that contain (at least) a world where d is 15, and the query returns the objects &7 and &52. The next step elaborates on the previous query, and returns addresses together with the contexts under which those addresses hold:

```
select address: <[Y]: X>
from [% d=15]music_club.address::[Y][-] X
```

The result is:

```
{address:
   <[d in {start..now}, season in {fall,winter,spring}]: &7 {...},
    [d in {10..now}, season=summer]: &52 {...}>
}
```

The query uses the context variable [Y] to return the contexts that correspond to address facets. For the next step, suppose that we are not interested in getting back dimension d, which we have already specified in the from clause. The query below returns the same results as before, but with contexts containing dimension season only:

```
select address: <[Z]: X>
context [Z] as [Y] * [% season in ALL]
from [% d=15]music_club.address::[Y][-] X
```

The result is:

```
{address:
   <[season in {fall,winter,spring}]: &7 {...},
    [season=summer]: &52 {...}>
}
```

The context clause uses the context pattern [% season in ALL] to screen out dimensions other than season in contexts bound to [Y]. The above query demonstrates the value and versatility of context patterns. To amplify this, observe the use of context patterns in the following query. It returns addresses that hold exactly from time instance 10 until now, irrespective of the values of other dimensions:

```
select address: X
from [Y]music_club.address X
within [Y] = [% d in {10..now}]
```

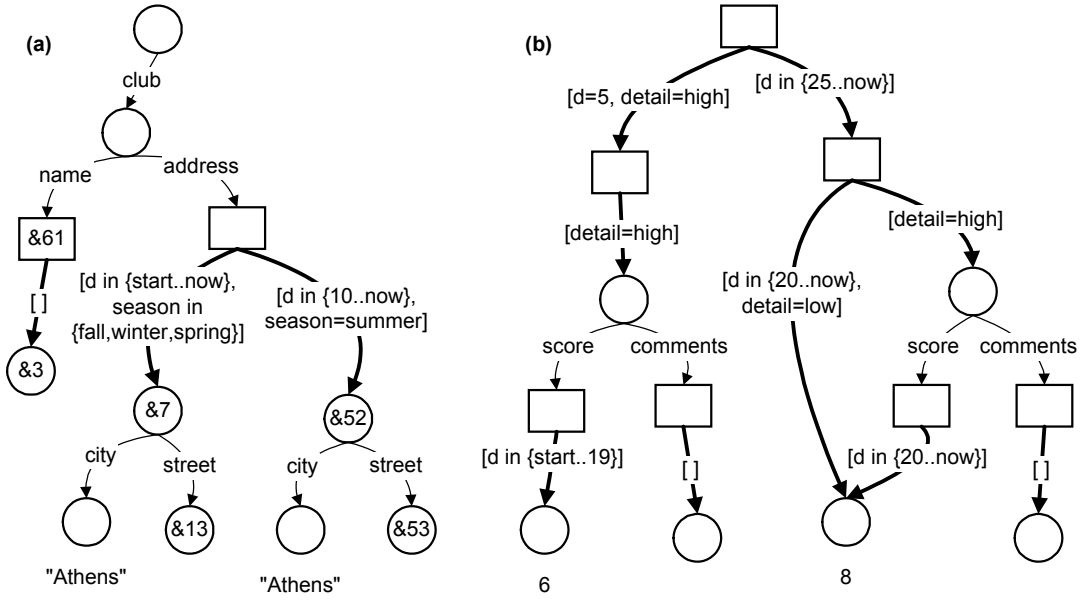
The query returns the object &52.

As another example, the query below retrieves the current address and the name of clubs, which at time instance 6 were located in Athens with zipcode 17671:

```
select club: {name: X.name, address: <[Q]: Z>}
from music_club X,
     X.[% d=6]address Y,
     X.[% d=now]address::[Q][-] Z
where Y.city = "Athens" and Y.zipcode = 17671
```

The result of the query is depicted in Figure 6.10 (a), and shows that a club that at time instance 6 was located in Athens with zipcode 17671 has currently two addresses, one for the summer and one for the rest of the year.

Figure 6.10: Results of MQL queries on MOEM histories.



The query that follows returns the reviews that hold under some time instance between 12 and 22, together with their actual valid time, irrespective of other dimensions:

```
select [Z]: X
context [Z] as [Y] * [% d in ALL]
from [Y]music_club.review X
within [Y] * [% d in {12..22}] != [-]
```

The result is:

```
<[d in {start..19}]: &8 6,
 [d in {20..now}]: &55 8,
 [d in ALL]: &9 {score: &54 <...>, comments: &62 <...>}
>
```

The result encompasses all `review` facets, because they all hold under some time instance between 12 and 22. Notice that only the dimension `d` appears in the contexts in the result, because other dimensions – namely `detail` – have been screened out by the context pattern `[% d in ALL]`. Once more, this query demonstrates how context patterns can manipulate contexts that involve many dimensions in an elegant and powerful way.

As a final example, the query below reduces partially the `review` multidimensional object for two contexts.

```
select holding <[d=5,detail=high]: <X>, [d in {25..now}]: <X>>
from music_club.review <X>
```

Figure 6.10 (b) shows the result graph. The multidimensional object variable `<X>` binds to node &5, and the keyword `holding` causes nodes and edges that are irrelevant to the reduction context to be removed throughout the subgraph. The partial reduction for `[d=5,detail=high]` gives the review contents in high detail at time instance 5, while the partial reduction for `[d in {25..now}]` gives the review contents in any detail that hold under some time instance between 25 and `now`.

6.6 RELATED WORK

The ability to model the temporal dimension of the real world is essential to many computer applications. Over the past two decades there has been a substantial amount of research on extending databases to support time [SA85, JCG+92, ZCF+97, OS95, TG95, BCW93, Org96]. In addition, some recent research investigates ways for incorporating time in semistructured data [CAW99, OQT01, DOQT01, GS98] and XML [GS98, AYU00, GM00, MGSI01a, Dyr01, Nor02, Gra02, SD02].

The problem of representing and querying changes in semistructured data has been studied in [CAW99, CAW98], where Delta OEM (DOEM in short) has been proposed. DOEM is a graph model that extends OEM with annotations containing temporal information. Four basic change operations, namely *creNode*, *updNode*, *addArc*, and *remArc* are considered by the authors in order to modify an OEM graph. Those operations are mapped to four types of annotations. Annotations are tags attached to a node or an edge, containing information that encodes the history of changes for that node or edge. When an OEM basic change operation takes place, a new annotation is added to the affected node or edge, stating the type of the operation, the timestamp, and in the case of *updNode* the old value of the object. The modifications suggested by the basic change operations actually take place, except from edge removal, which results to just annotating the edge. To query DOEM databases, the query language Chorel is proposed. Chorel extends Lorel with constructs called *annotation expressions*, which are placed in path expressions and are matched against annotations in the DOEM graph. Annotation expressions may contain *time variables* that allow stating conditions on timestamps. Chorel deals with single time instances and time intervals by introducing special keywords in annotation expressions, like `at`, `during`, and `snap`.

Our approach is based on some of the key concepts presented in [CAW99]. It is however quite different, as changes are represented by introducing new facets instead of adding annotations. Similar to DOEM, MOEM can give the snapshot of an OEM database at any

time instance, through the process of reduction to OEM. The principal difference between the query languages stems, as one would expect, from the different representations of OEM histories. Chorel, with its annotation expressions, maintains at a central role the notion of OEM basic change operations, which apply to a specific element and occur at a specific time instance. Then, starting from this basis, Chorel builds higher-level concepts, like for example an element holding under a time interval, or a complete path holding under a time instance. On the other hand, the notion of OEM basic change operations does not exist in MQL. MQL queries are addressed to the results of those operations, kept as facets of multidimensional entities. The central notion in MQL is context, which means that MQL directly handles the valid time intervals of database elements and database paths.

A special graph for modeling the dynamic aspects of semistructured data, called semistructured temporal graph is proposed in [OQT01]. In this graph, every node and edge has a label that includes a part stating the valid interval for the node or edge. Modifications in the graph cause changes in the temporal part of labels of affected nodes and edges.

An approach for representing temporal XML documents is proposed in [AYU00], where leaf data nodes can have alternative values, each holding under a time period. However, the model presented in [AYU00] does not allow dimensions other than time, and does not explicitly support facets with varying structure for nodes that are not leaves. Other approaches for representing time in XML include [GM00, MGSI01a, Gra02]. In [Dyr01] the XPath data model is extended to support transaction time. The query language of XPath is extended as well with transaction time axis to enable to access past and future states. Constructs that extract and compare times are also proposed. Finally, in [SD02] the XPath data model and query language is extended to include valid time, and XPath is extended with an axis to access valid time of nodes.

An important advantage of MSSD over other approaches for managing SSD histories is that MSSD can be applied to a variety of problems from different fields; representing valid time is just one of the possible applications of MSSD. MOEM is suitable for modeling entities that present different facets, a problem often encountered on the Web, and the representation of semistructured database histories can be seen as a special case of this problem. Properties and processes defined for the general case of MSSD (like path inherited coverage, reduction to OEM, partial reduction) are also used without change in the case of representing semistructured histories. Similarly, MQL is not made especially for querying histories of semistructured databases, but is targeted at the general case of context-dependent semistructured data. Therefore, the concepts we proposed as part of MOEM and MQL have a wider applicablility, and are not confined in the frame of a specific problem. Moreover, as we showed in Sections 6.4 and 6.5.2, MOEM and MQL are capable of representing and querying MOEM's own histories, without having to introduce any additional concepts.

6.7 SUMMARY

In this chapter we explained in depth how Multidimensional OEM, a graph model for context-dependent semistructured data, can be used to represent the history of an OEM database. We introduced the **basic change operations** of MOEM, and specified in pseudocode the process that encodes in an MOEM the history of an OEM database. We discussed how Multidimensional Data Graph properties apply in this particular case, and showed that temporal OEM snapshots can be obtained from MOEM. We presented *OEM History*, an implemented system, and demonstrated through an example the process of using an underlying MOEM database to model changes in OEM. In addition, we showed that

MOEM is capable to model changes occurring not only in conventional OEM databases, but in MOEM databases as well.

An important benefit of our approach stems from the queries that can be posed on OEM histories and MOEM histories. Our approach maintains snapshots of the successive states of an object as facets of a multidimensional entity. This allows the formulation of queries that relate different states of the same or different object(s) at various time instances. We presented a number of MQL query examples, and through them we confirmed the expressiveness of MQL and the value of some of its features, like for example context patterns.

In our view, this example application of MOEM and MQL demonstrates the potential of MSSD. The applicability of MOEM and MQL is not exhausted in representing histories of semistructured data. Context-dependent data are of increasing importance in a global environment such as the Web, and MSSD can be applied to problems from disperse domains, among which: in delivery of personalized information and services; in information integration where objects vary in value and structure according to sources; in representing and querying geographical information where possible dimensions are *scale* and *theme*.

7 MULTIDIMENSIONAL XML

EXtensible Markup Language, or XML in short, is a markup language that has become a standard for data representation and exchange over the Web [XML, Wal97, Cha99]. XML resembles Hypertext Markup Language [HTML], or HTML in short, but unlike HTML it focuses on the structure of data rather than on presentation. XML can be seen either from a document-centric perspective, or a data-centric one. The document-centric view originates from SGML [SGML], the markup language that inspired the design of XML, and sees XML as a way to embed in Web documents information about their structure. The data-centric perspective perceives XML as a language suitable for expressing semistructured data and for exchanging data over the Web. From this perspective, the emphasis is on querying and on describing the relationships between pieces of data, in a way similar to a database schema.

Although the main characteristic of XML is its extensibility in terms of defining new element types at will, it falls short when it comes to representing information that display different facets under different contexts. As a simple example, imagine a report that needs to be represented at various degrees of detail, and in various languages. A solution would be to create a different XML document for every possible combination of variations, and attach a kind of "label" indicating the context of each document. Such an approach is certainly not practical, since it involves excessive duplication of information. What is more, different facets in different documents are not associated as being parts of the same entity.

Ideas on how this problem can be handled are given in [SGR00, SGM00], where **Multidimensional XML**, or **MXML** in short, is presented. Moreover, some preliminary thoughts of representing time using MXML are discussed in [MGSI01, MGSI01a]. In this chapter, we consolidate work published in [GSK01, GSK+01]. Specifically, we describe how MXML extends the syntax of XML, by allowing context specifiers to qualify elements and attributes and specify the worlds under which document components have meaning. We also propose a **mutlidimensional paradigm** for viewing context-dependent Web data, and we present a system that implements the basic functionality of this multidimensional paradigm.

MXML was influenced by Intensional HTML [WBSY98, Bro98, Yil97, Bro98a], or IHTML in short, a Web authoring language that is based on and extends ideas proposed for a software versioning system [PW93]. IHTML allows a single Web page to have different variants and to dynamically adapt itself to a given context. The main difference between MXML and IHTML is a projection of the main difference between XML and HTML, i.e. the emphasis is on encoding the structure rather than on the presentation of data.

The idea of incorporating multiple dimensions in languages is not new. GLU [AFJ91, AFJW95] is a functional programming language that allows the user to declare dimensions and to define entities that vary across these dimensions. Moreover, a language in the spirit of GLU has been developed in the domain of logic programming [OD97], while ISE [SW00] is a multidimensional version of Perl [PERL].

Our work on MSSD started as an attempt to incorporate context in XML, however we soon turned to OEM as a simpler and more abstract basis on which to tackle emerging

problems. In what follows *we do not present MXML as a completed work, but rather as an example of how the ideas introduced in previous chapters can expand to other directions*.

7.1 INCORPORATING CONTEXT IN XML

In a **multidimensional XML document**, or **MXML document** in short, context may qualify elements and attributes. An element whose content depends on one or more dimensions is called **multidimensional element**, while an attribute whose value depends on one or more dimensions is called **multidimensional attribute**.

In this section, we introduce MXML as an extension of the XML syntax that encompasses context specifiers and can represent multidimensional entities. We focus on this part of documents that corresponds to the logical structure of information, and we do not address issues related to the physical structure of XML documents. Moreover, we assume that MXML retains the XML syntax for referencing DTD and XSL files.

7.1.1 Why Extend the XML Syntax?

Someone keen on the preservation of standards may view the fact that MXML modifies the syntax of XML as a disadvantage. The alternative approach would be to define MXML as a conventional DTD (an XML "instance"), without extending the XML syntax, and use namespaces [XNS] to combine it with domain-specific DTDs. There are three reasons why we have chosen to extend the syntax of XML:

(a) Context can be used in a wide variety of unrelated domains, so in our view Web data models and languages should support context as a first-class citizen. By defining MXML as a conventional DTD we feel that context is demoted, and that its role and potential in the Web is underestimated.

(b) XML could in fact represent multifaceted *elements* with the help of a few reserved symbols, defined in a conventional DTD. However, in order to express multidimensional *attributes* without extending the XML syntax, attributes should be turned into a special kind of XML elements. The problem with this approach is that a "context-aware" XML becomes even more verbose than XML already is, and – although understandable by a machine – unsuitable for demonstrating to other humans the essential point. The essential point, which is context and multifaceted information, is hidden behind the difficult interpretation of a weird syntax.

(c) As explained in Chapter 2, our view of XML is that of a logical model for data rather than a physical format. Supporting context explicitly at this level would facilitate the use of context-aware query languages for XML. Context-driven queries are easier to formulate if they use concepts that correspond directly to the underlying data model.

The above reasons seemed adequate for us to extend the XML syntax. However, once the validity of context-aware multidimensional data models and languages is established, *it may be reasonable for practical purposes to express multifaceted information through conventional XML*, instead of using an XML extension like MXML.

7.1.2 Multidimensional XML Syntax

The syntax of XML is extended as follows in order to incorporate context specifies.

- A multidimensional element has the form:

```
<@element_name  attribute_specification>
    [context_specifier_1]
        <element_name attribute_specification_1>
            element_content_1
        </element_name>
    [/]
    . . .
    [context_specifier_N]
        <element_name attribute_specification_N>
            element_content_N
        </element_name>
    [/]
</@element_name>
```

A multidimensional element is denoted by preceding the element name with the special symbol `@`, and encloses one or more **context elements** that constitute facets of that multidimensional element, holding under the worlds specified by the corresponding context specifier. Context elements have the same form as conventional XML elements. All context elements belonging to a multidimensional element have the same name, which is the name of the multidimensional element.

- A multidimensional attribute has the form:

```
attribute_name = [context_specifier_1] attribute_value_1 [/]
                 . . .
                 [context_specifier_N] attribute_value_N [/]
```

A multidimensional attribute is expressed as an attribute whose value is a set of context / value pairs. Each one of those context-qualified values becomes the holding value of the attribute for the corresponding context.

Example 7.1 shows part of an MXML document describing the imaginary multidimensional menu of a restaurant.

Example 7.1: An MXML describing the menu of a restaurant.

```
<restaurant>
  <menu>
    <salad name="Chef's salad" vegetarian=[season=summer]
                                            "yes"[/]
                                          [season!=summer]
                                            "no"[/]>

      <@comment>
        [language=English, detail=low]
          <comment>A traditional salad.</comment>
        [/]
        [language=English, detail=high]
          <comment>
            A salad with a long history, which
            has roots in the tradition of the town.
          </comment>
        [/]
        [language=French, detail in {low,high}]
          <comment>Une salade regionale traditionelle.</comment>
        [/]
      </@comment>
```

```
      <@price>
        [season=summer] <price>3 EUR</price> [/]
        [default] <price>4 EUR</price> [/]
      </@price>

      <ingredient>tomato</ingredient>

      <@ingredient>
        [season!=summer] <ingredient>bacon</ingredient> [/]
      </@ingredient>

      <ingredient>olive oil</ingredient>

      <@ingredient>
        [occasion=special]
          <ingredient special_supplier =
                          [season in {spring,summer}] "sp1" [/]
                          [default] "sp2" [/]>
            <name>special sauce</name>
            <remarks>Must order three days in advance</remarks>
          </ingredient>
        [/]
        [default] <ingredient>normal sauce</ingredient> [/]
      </@ingredient>

      ... other salad ingredients ...

    </salad>

    ... other menu items ...

  </menu>

  <supplier scode="sp1">
    <name>John Smith</name>
    <address>234 XYZ Street</address>
  </supplier>

  <supplier scode="sp2">Peter Brown</supplier>

<\restaurant>
```

♦

Context specifiers are integral parts of the context elements they qualify. In addition, the `[default]` context specifier is used as shorthand for all worlds not covered by other context specifiers in the same multidimensional entity. As in Multidimensional Data Graph, the root of an MXML document can be a conventional element, or a multidimensional element.

7.1.3 Dimensions Applied to Elements

Multidimensional elements can only contain context elements[109], while context elements may contain multidimensional elements, conventional elements[110], or any combination of the

[109] In Multidimensional Data Graph multidimensional nodes can point to multidimensional nodes. In MXML a multidimensional element cannot contain another multidimensional element; only multidimensional attributes can be used to reference multidimensional elements.

two in an arbitrary depth. Context elements of the same multidimensional element are not required to have the same content, or even to conform to the same structural constraints as they could be defined in a DTD. Therefore, dimensions can affect the content of an element in every aspect, be it its structure or its value.

The effect of context in element value: Consider the multidimensional element `comment` in Example 7.1, whose value depends on dimensions `language` and `detail`. The context specifier of the third context element of `comment` is `[language=French, detail in {low, high}]`. As a result, the value of `comment` under `[language=French, detail=low]` and `[language=French, detail=high]` is "`Une salade regionale traditionelle`".

The effect of context in element structure: Context specifiers also affect the element structure. As an example, consider the fourth ingredient of the salad in Example 7.1, where the element `ingredient` contains the subelements `name` and `remarks` when its context is `[occasion=special]`, but for all the other values of the dimension `occasion` (denoted by the context specifier `[default]`), it contains no subelements.

Notice that it is not necessary for a multidimensional element to have context elements for every possible world. For example, the multidimensional element

```
<@ingredient>
    [season!=summer] <ingredient>bacon</ingredient> [/]
</@ingredient>
```

in Example 7.1, has no facet for the context `[season=summer]`.

There is a case where a multidimensional element or attribute is equivalent to a conventional element or attribute. For example, consider the element `ingredient` in Example 7.1:

```
<ingredient>tomato</ingredient>
```

The above is in fact shorthand for:

```
<@ingredient>
    [] <ingredient>tomato</ingredient> [/]
</@ingredient>
```

7.1.4 Dimensions Applied to Attributes

In an XML document, a piece of data expressed as an attribute can be also expressed as an element. The reverse is not true, since attributes cannot be of complex type. Actually, from the perspective of expressing multidimensional data, the only important difference between elements and attributes is the use of attributes for associating elements through the "ID", "IDREF", and "IDREFS" attribute types, which can be seen as analogous to the element – subelement relationship. Therefore, elements and attributes should be considered conceptually equivalent, and whatever holds for elements should in principle hold for attributes.

---

[110] In MXML context elements and conventional elements have the same form, nevertheless they are considered as different categories because they have different MDTD declarations, as we will see in Section 7.2.2.

Within a multidimensional element, each context element can have its own attributes (conventional or multidimensional), perhaps different from those of other context elements, exactly as it can have its own child elements.

Context specifiers determine which are the attributes of an element under a world. Notice that in Example 7.1, a salad `ingredient` has the attribute `special_supplier` for the context `[occasion=special]`, but for any other value of `occasion` (denoted by `[default]`) the element `ingredient` has no attributes.

Attributes of type "ID" can be attached to multidimensional elements[111] as well as to context and conventional elements, while attributes of type "IDREF" and "IDREFS" can only be attached to context and conventional elements. Thus, by using attributes of types "IDREF" and "IDREFS", context and conventional elements are able to reference (or "point" to) multidimensional, conventional, or context elements. Actually, the only attribute that can be attached to multidimensional elements is of type "ID". Hence, although multidimensional elements can be pointed to by "IDREF" attributes, they cannot themselves use attributes to point to other elements[112]. Attributes of type "ID" cannot be multidimensional attributes.

The attribute `special_supplier` in the fourth `ingredient` element of the Example 7.1, has value `"sp1"` under the worlds covered by `[season in {spring,summer}]`, and value `"sp2"` under every other world (represented by `[default]`). Notice that, as the value of this attribute is a reference pointing to a `supplier` element, this attribute points to different `supplier` elements depending on the values of the dimension `season`. As another example, consider the multidimensional attribute `vegetarian` of the element `salad`, which tags the salad as suitable for vegetarians or not depending on the dimension `season`.

7.2 MULTIDIMENSIONAL DTD

An XML Document Type Definition [XML], or DTD in short, is used as a specification that defines constraints on the logical structure of XML documents. In this section, we propose an extension of DTD called **Multidimensional DTD**, or **MDTD** in short, which takes context and dimensions into account, and describes the logical structure of MXML documents. An MXML document is **valid** with respect to an MDTD if it conforms to the constraints declared in that MDTD.

7.2.1 Dimension Declarations

Dimensions are declared in MDTD as:

```
<!DIMENSION dimension_name dimension_domain>
```

Using dimension declarations we can declare a dimension and associate it with the set of possible values. For example, the declaration

```
<!DIMENSION language {English,French}>
```

[111] ID attributes of multidimensional elements are used only for referencing and should not convey any information (like, for instance, the ISDN of a book), because they do not survive under any world.

[112] A context element in MXML is pointed to only by the multidimensional element that contains it. Nevertheless, it can be pointed to by many multidimensional attributes. This is similar to MOEM, where a node can be pointed to by any number of multidimensional nodes.

denotes that `language` is a dimension name, and defines its domain as the set `{English,French}`. Alternatively, it is possible to declare the dimension domain using a URI, as in

```
<!DIMENSION branch_no "http://company_server/MXML/branches.html">
```

7.2.2 Element Declarations

The conventional DTD syntax is used in MDTD for declaring conventional elements. Multidimensional element declarations deal with both multidimensional elements and context elements. Multidimensional element declarations are of the form:

```
<!MULTIELEMENT element_name associated_dimensions type_decl>
```

The set of dimensions on which the multidimensional element depends replaces `associated_dimensions`. For example, in the following declaration

```
<!MULTIELEMENT comment {language,detail} (#PCDATA)>
```

the element `comment` is declared to be a multidimensional element that depends on dimensions `language` and `detail`.

Using multidimensional element declarations, it is possible to define varying structural constraints on the context elements of a multidimensional element. As an example, in the following declaration

```
<!MULTIELEMENT ingredient {season,occasion}
    [occasion=special] ((name,remarks?) | #PCDATA) [/]
    [default] (#PCDATA) [/]
>
```

the type of the element `ingredient` is declared to be `(name,remarks?) | #PCDATA` whenever the value of the dimension `occasion` is `special`; under any other world, the type of the element `ingredient` is declared to be `(#PCDATA)`.

7.2.3 Attribute Declarations

Attribute declarations have been extended to take into account multidimensional attributes. For example, in the declaration

```
<!ATTLIST salad
    name CDATA #REQUIRED
    vegetarian {season} CDATA #IMPLIED
>
```

the element `salad` is declared to have two attributes, namely `name` and `vegetarian`. The value of the attribute `name` does not depend on dimensions, while the value of the attribute `vegetarian` depends on the dimension `season`.

Attribute declarations allow declaring that an attribute is present under some contexts, while it is absent under other contexts. For example, in the declaration

```
<!ATTLIST ingredient
  [occasion=special] special_supplier {season} IDREF #REQUIRED [/]
>
```

the element `ingredient` is declared to have the attribute `special_supplier` only under the worlds in which the value of the dimension `occasion` is `special`. In all other worlds the element `ingredient` has no attributes.

The declaration

```
<!ATTLIST @element_name attribute_name ID>
```

uses the character `@` to declare an attribute of type "ID" for a multidimensional element.

7.2.4 An MDTD Example

In Example 7.2 we present an MDTD for the MXML document of Example 7.1.

Example 7.2: An MDTD for the MXML of Example 7.1.

```
<!DOCTYPE menuDTD [
<!DIMENSION language {English,French}>
<!DIMENSION detail   {low,high,exhaustive}>
<!DIMENSION season   {spring,summer,fall,winter}>
<!DIMENSION occasion {special,normal}>

<!ELEMENT restaurant (menu | supplier)*>
<!ELEMENT menu  (salad+, first+, maindish+, dessert+)>
<!ELEMENT salad (comment?, price, ingredient*)>
<!ATTLIST salad
          name CDATA #REQUIRED
          vegetarian {season} CDATA #IMPLIED>

<!MULTIELEMENT comment {language,detail} (#PCDATA)>
<!MULTIELEMENT price {season} (#PCDATA)>
<!MULTIELEMENT ingredient {season,occasion}
          [occasion=special] ((name, remarks?) | #PCDATA) [/]
          [default] (#PCDATA) [/]>
<!ATTLIST ingredient
          [occasion=special]
              special_supplier {season} IDREF #REQUIRED
          [/]>
<!ELEMENT name     (#PCDATA)>
<!ELEMENT remarks  (#PCDATA)>

<!ELEMENT supplier (name, address)>
<!ATTLIST supplier
          scode ID #REQUIRED>
]>
```

♦

The MDTD of Example 7.2 declares that the document in Example 7.1 depends on four dimensions: the dimension `language` whose domain is {`English,French`}, the dimension `detail` whose domain is {`low,high,exhaustive`}, the dimension `season` whose domain is {`spring,summer,fall,winter`}, and the dimension `occasion` whose domain is {`special,normal`}. Besides conventional elements such as `menu` and `salad`, the multidimensional elements `comment`, `price`, and `ingredient` are declared.

7.3 PROPERTIES OF MXML

It is evident that MXML documents can be turned into Multidimensional Data Graphs, by considering MXML multidimensional elements and multidimensional attributes as corresponding to multidimensional nodes. MXML attributes of type "IDREF" and "IDREFS" are represented in the Multidimensional Data Graph as edges pointing to the referenced object: entity edges for conventional attributes, and context edges for multidimensional attributes[113].

Multidimensional Data Graph and MOEM are more flexible in representing MSSD than MXML. Due to its XML document-centric origins, the syntax of MXML impose limitations not encountered in Multidimensional Data Graph and MOEM. For example, it is not possible to represent in MXML a path of three or more consecutive context edges[114]. Nevertheless, a Multidimensional Data Graph can always be expressed in MXML if we consider its canonical form.

Properties of Multidimensional Data Graph apply to MXML as well. The concepts of explicit context, inherited context, context coverage, inherited coverage, path inherited coverage, and context determinism are the same in MXML as in Multidimensional Data Graph. In addition, the processes of reduction to OEM and partial reduction can be adjusted for MXML. Each MXML document represents in fact a set of conventional XML documents. Given a world w, an MXML document can be reduced to a conventional XML document, which is the facet of the multidimensional document under w. Example 7.3 demonstrates **reduction to XML**, and shows the conventional XML facet of the MXML in Example 7.1 that holds under the world {(language,English), (detail,low), (season,summer), (occasion,special)}.

Example 7.3: An XML facet of the MXML in Example 7.1.

```
<restaurant>
  <menu>
    <salad name="Chef's salad" vegetarian="yes" >
      <comment>A traditional salad.</comment>
      <price>3 EUR</price>
      <ingredient>tomato</ingredient>
      <ingredient>olive oil</ingredient>
      <ingredient special_supplier="sp1" >
        <name>special sauce</name>
        <remarks>Must order three days in advance</remarks>
      </ingredient>

      ... other salad ingredients ...

    </salad>

    ... other menu items ...

  </menu>
```

[113] Note that in order to be able to distinguish "IDREF" and "IDREFS" attributes from other attributes, we need to consult the corresponding MDTD, where attribute types are declared.

[114] Two consecutive context edges can be represented as a multidimensional attribute referencing a multidimensional element.

```
    <supplier scode="sp1">
      <name>John Smith</name>
      <address>234 XYZ Street</address>
    </supplier>

    <supplier scode="sp2">Peter Brown</supplier>

  <\restaurant>
  ♦
```

The notion of **well-formed** MXML is an extension of the notion of well-formed XML [XML]. The XML well-formedness criteria hold for elements and attributes of MXML as well. In addition, a well-formed MXML document must: (a) be context-deterministic, and (b) contain exclusively elements and attributes that have non-empty inherited coverage. Therefore, *the Multidimensional Data Graph that represents a well-formed MXML is an MOEM*.

7.4 THE MULTIDIMENSIONAL PARADIGM

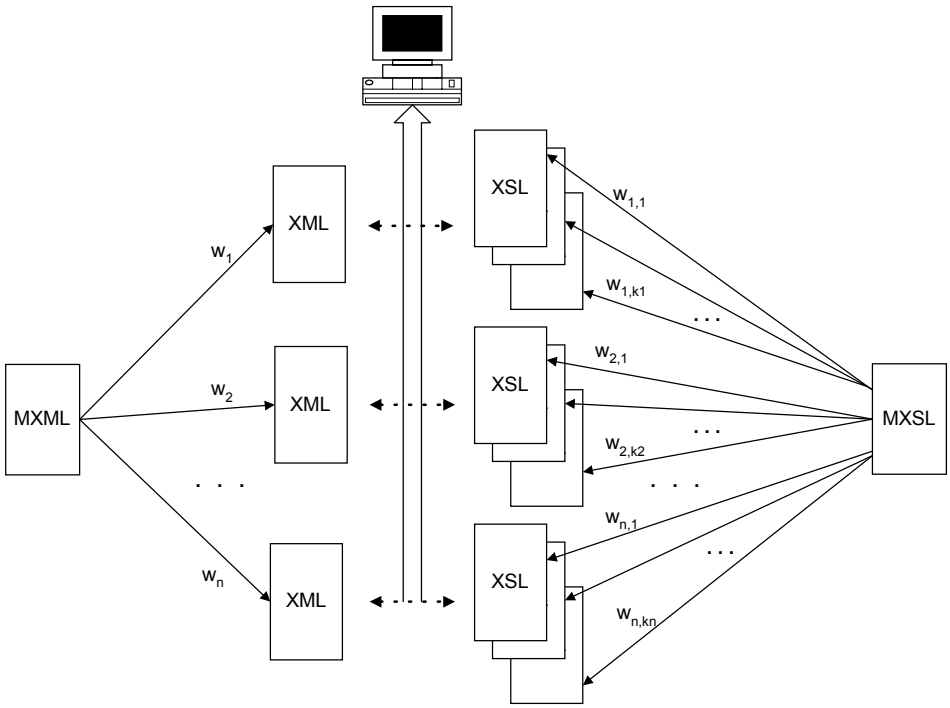
Web information entities that may assume different facets under different contexts lead to a new paradigm for manipulating and viewing context-dependent Web data. We refer to the new paradigm as the **multidimensional paradigm**. In this section, we explain the paradigm we propose, and use MXML documents and **multidimensional XSL stylesheets** to demonstrate its use through a comprehensive example.

7.4.1 Viewing Multidimensional SSD

XML does not address the issue of how to present information to the user. EXtensible Stylesheet Language [XSL], or XSL in short, offers a solution to this. XSL is a document containing instructions on how to present information in XML documents. In addition, it is possible to select pieces of data from an XML document and to transform them to another format (for instance, HTML), using XSL Transformations [XSLT] (XSLT in short). It is important to note that XSL stylesheets are also XML documents, and they conform to the XML syntax. An XSL stylesheet can be applied to a specific XML document, and the result can be displayed by a Web browser. Actually, a number of XSL documents can be applied to the same XML document, giving different ways to view the document or parts of it.

The multidimensional paradigm allows a single document to have a number of facets, each holding under different worlds. Information in such a document is encoded in a suitable markup language, such as MXML. Once a world is specified, an MXML document can be reduced to a conventional XML document that constitutes the facet under that world. This decomposition of an MXML document to a number of conventional XML facets is depicted in Figure 7.1, where worlds are denoted as w_1, w_2, ..., w_n.

Figure 7.1: MXML, MXSL, and their relation.



Since XSL stylesheets are XML documents, the same principles hold for XSL stylesheets as well. A **multidimensional XSL stylesheet**, or **MXSL** in short, encodes a set of conventional XSL stylesheets, each being the facet of the MXSL under a specific world. Like a conventional XSL, an MXSL must be associated with an XML or an MXML document. For each possible world, the holding XSL facet is applied to the holding XML facet to display the document under that world. The relation between MXML and MXSL is given pictorially in Figure 7.1. Note that the possible worlds for an MXSL may not be identical to those of the corresponding MXML. A number of additional dimensions in an MXSL may be used to define alternative presentations for the same XML document. On the other hand, some of the dimensions in an MXML document may appear in the corresponding MXSL as well, in order to establish a correspondence between the holding MXML and MXSL facets.

7.4.2 A Comprehensive Example

Example 7.4 shows an MXML document about a book that exists in two different editions, an English and a Greek one. The element `book` has six subelements, described next. The `isbn` and `publisher` are multidimensional elements, and depend on the dimension `edition`. The elements `title` and `authors` remain the same under every possible world. The element `price` is a multidimensional element whose value depends on the dimensions `edition` and `customer_type`. Finally, the element `translator` has substance only under the worlds where `edition` has the value `greek`.

Example 7.4: An MXML containing multidimensional information about a book.

```
<book>
  <@isbn>
      [edition=greek] <isbn>0-13-110370-9</isbn> [/]
      [edition=english] <isbn>0-13-110362-8</isbn> [/]
  </@isbn>

  <title>The C programming language</title>

  <authors>
      <author>Brian W. Kernighan</author>
      <author>Dennis M. Ritchie</author>
  </authors>

  <@publisher>
      [edition=english] <publisher>Prentice Hall</publisher> [/]
      [edition=greek] <publisher>Klidarithmos</publisher> [/]
  </@publisher>

  <@translator>
      [edition=greek] <translator>Thomas Moraitis</translator> [/]
  </@translator>

  <@price>
      [edition=english,customer_type=individual]
                                          <price>38</price> [/]
      [edition=english,customer_type=library]
                                          <price>29.4</price> [/]
      [edition=english,customer_type=student]
                                          <price>34</price> [/]
      [edition=greek,customer_type=individual]
                                          <price>15</price> [/]
      [edition=greek,customer_type=library]
                                          <price>8.8</price> [/]
      [edition=greek,customer_type=student]
                                          <price>13</price> [/]
  </@price>
</book>
```

♦

The MXML in the following Example 7.5 is an MXSL stylesheet for the MXML document of Example 7.4, and specifies how the various facets of the MXML of Example 7.4 are to be presented.

Example 7.5: An MXSL for the MXML of Example 7.4.

```
<xsl:template match="/">
  <DIV STYLE=[size=large]"font-size:22pt"[/]
             [size=normal]"font-size:18pt"[/]>
    Book
  </DIV>

  <@SPAN>
    [customer_type=library]
      <SPAN STYLE="font-size:15pt">
        ISBN: <xsl:value-of select = "book/isbn"/>,
      </SPAN>
    [/]
```

```
    [customer_type in {individual,student}]
      <SPAN STYLE="font-size:15pt">
        Title: <xsl:value-of select="book/title"/>,
        Authors: <xsl:value-of select="book/authors"/>,
        <@wrapper>
          [edition=greek]
            <wrapper>
              Translator: <xsl:value-of select="book/translator"/>,
            </wrapper>
          [/]
        </@wrapper>
      </SPAN>
    [/]
  </@SPAN>

  <SPAN STYLE="font-size:15pt">
    Publisher: <xsl:value-of select="book/publisher"/>,
  </SPAN>

  <SPAN STYLE="font-size:15pt">
    Price: <xsl:value-of select="book/price"/>
  </SPAN>
</xsl:template>
```

♦

The `ISBN` is shown only if the request for information has been made by a library, while `title` and `authors` are shown if the potential client is an individual or a student. Notice how the `wrapper` element, which is defined in XSL[115], can also be used elegantly in MXSL to exclude the `translator` in case the request concerns the original language edition of the book. In general, MXSL may contain multidimensional versions of elements and attributes defined for conventional XSL. Finally, the elements `publisher` and `price` are displayed in any case.

There is no constraint on which dimensions appear in the context specifiers of an MXSL. Some of them may also appear in the corresponding MXML document, as is the case of dimensions `customer_type` and `edition` in Example 7.5, or they can be different, as is the case of dimension `size` in the same example, which does not exist in the MXML.

Under the world w = {`(edition,greek)`, `(customer_type,student)`, `(size,large)`}, the MXML document of Example 7.4 is reduced to the conventional XML document shown in Example 7.6. The fact that dimension `size` does not appear in the MXML of Example 7.4 does not present any problem, since the complete domain of `size` is implied in the context specifiers.

Example 7.6: An XML facet of the MXML in Example 7.4.

```
<book>
  <isbn>0-13-110370-9</isbn>
  <title>The C programming language</title>
  <authors>
    <author>Brian W. Kernighan</author>
    <author>Dennis M. Ritchie</author>
  </authors>
  <publisher>Klidarithmos</publisher>
  <translator>Thomas Moraitis</translator>
```

[115] As specified in XSL, the `wrapper` element has no semantics but acts as a "carrier" of formatting properties, which are applied to its contents.

```
  <price>13</price>
</book>
```

♦

Under the same world w, the MXSL stylesheet of Example 7.5 is reduced to the XSL stylesheet shown in Example 7.7.

Example 7.7: An XSL facet of the MXSL in Example 7.5.

```
<xsl:template match="/">
  <DIV STYLE="font-size:22pt">Book</DIV>
  <SPAN STYLE="font-size:15pt">
      Title: <xsl:value-of select="book/title"/>,
      Authors: <xsl:value-of select="book/authors"/>,
      <wrapper>
          Translator: <xsl:value-of select="book/translator"/>,
      </wrapper>
  </SPAN>
  <SPAN STYLE="font-size:15pt">
      Publisher: <xsl:value-of select="book/publisher"/>,
  </SPAN>
  <SPAN STYLE="font-size:15pt">
      Price: <xsl:value-of select="book/price"/>
  </SPAN>
</xsl:template>
```

♦

The result of applying the XSL stylesheet of Example 7.7 to the XML document of Example 7.6 looks like this:

Book

Title: The C programming language, Authors: Brian W. Kernighan, Dennis M. Ritchie, Translator: Thomas Moraitis, Publisher: Klidarithmos, Price: 13

Finally, the result of an analogous process for the world {`(edition,english)`, `(customer_type,library)`, `(size,normal)`} will look like this:

Book

ISBN: 0-13-110362-8, Publisher: Prentice Hall, Price: 29.4

7.5 A PROTOTYPE IMPLEMENTATION

In this section, we describe a prototype system that demonstrates the basic principles of the multidimensional paradigm we proposed. The implemented system is called *MXML Web Server*, and is a Web server capable of handling MXML and MXSL multidimensional data.

In a typical scenario, the user asks for an MXML document through a conventional Web browser, and is prompted to select values for each of the dimensions associated with the requested document. After the user has specified a world, the server sends the corresponding XML and XSL facets to be displayed by the browser. The user can change the values of

dimensions, and observe how different worlds are associated to different views of the same multidimensional document.

An important point is that, in the multidimensional paradigm *the reduction process of MXML and MXSL can take place at the server, the client, or both*. Our system implements this functionality at the server side mainly for reasons of compatibility with existing Web browsers. In the general case, however, *some of the dimensions can be considered at server side using partial reduction*, in order to eliminate irrelevant data and reduce the size of the response, while *the rest of the dimensions could be handled at client side*, for reasons of privacy or for minimizing the number of subsequent requests. This flexibility could be exploited when using MXML in domains like electronic commerce and user modeling.

7.5.1 Extending URL

URL [URL], which is short for Uniform Resource Locator, is the standard way to specify a resource available on the Internet. We extended the syntax of URL in order to enable it to handle multidimensional information. The extended URL has the form:

```
http://<host>:<port>/<path><context>?<search>
```

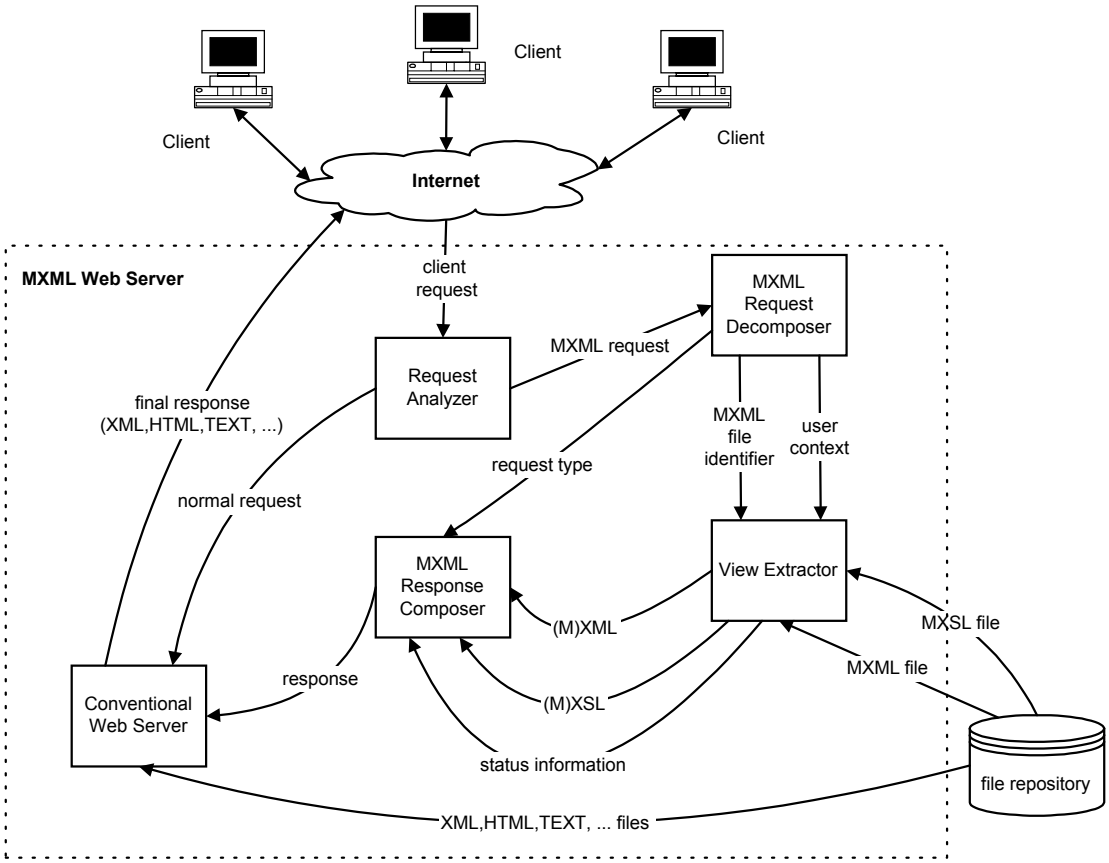
The only difference from conventional URL is that the token `<context>` has been added. The following extended URL is a request for the facet of a document named `books.mxml`, for which `edition` is `english` and `customer_type` is `student`.

```
http://myserver/books.mxml[edition=english,customer_type=student]
```

7.5.2 System Design

The system comprises the software modules illustrated in Figure 7.2. These modules are: *Request Analyzer*, *MXML Request Decomposer*, *View Extractor*, *MXML Response Composer*, and *Conventional Web Server*. In the following paragraphs we shall briefly describe the functionality of each of those modules, as well as their sub-modules, and discuss their role in the system operation.

Figure 7.2: *MXML Web Server* design.



Request Analyzer: The Request Analyzer is responsible for determining the type of client requests. There are two types of requests: *normal requests*, which refer to HTML, XML, text files etc., and *MXML-requests*, which refer to MXML files. If the request is of *normal* type, then the Conventional Web Server handles it. Otherwise, if the request involves an MXML document, then the MXML Request Decomposer is invoked.

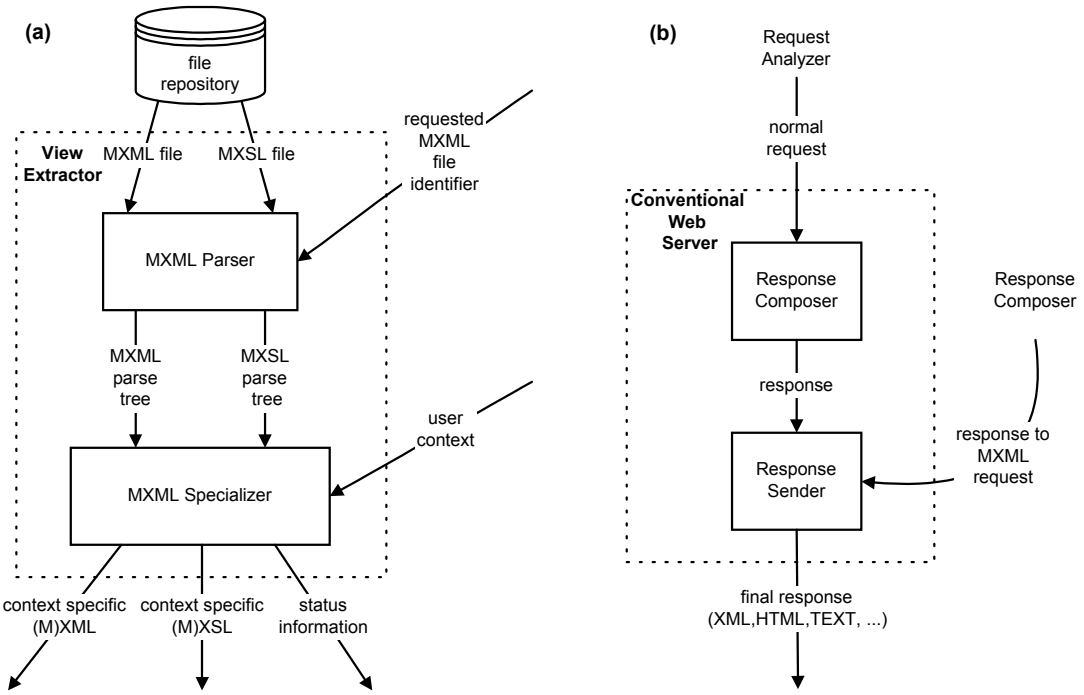
MXML Request Decomposer: This module decomposes the MXML request into sections. The three sections that comprise the MXML request are: the *MXML File Identifier*, the *User Context*, and the *Request Type*. The first two sections are sent to the View Extractor module, while the third is sent directly to the MXML Response Composer.

View Extractor: This module combines the information provided by the MXML Request Decomposer with the corresponding MXML or MXSL files, in order to serve the request. The View Extractor consists of two sub-modules, called *MXML Parser* and *MXML Specializer*, as is depicted in Figure 7.3 (a).

- *MXML Parser*: The MXML Parser uses the requested MXML File Identifier, which is provided by the MXML Request Decomposer, to access the corresponding MXML or MXSL file in the File Repository. Its role is to parse the file and generate the corresponding MXML or MXSL parse tree.
- *MXML Specializer*: The MXML Specializer uses the parse tree that is generated by the MXML Parser, plus the User Context, which is provided by the MXML Request Decomposer, to generate a context-specific (M)XML or (M)XSL file. Reduction to

XML is used in case User Context represents a single world, and partial reduction is used in case User Context represents more than one world. The type of the produced file is passed to *MXML Response Composer* through the *status information*.

Figure 7.3: Sub-modules of *View Extractor* are depicted in (a), and sub-modules of *Conventional Web Server* are depicted in (b).



MXML Response Composer: The MXML Response Composer constructs the response and sends it to the *Response Sender*, which dispatches the response to the client.

Conventional Web Server: The Conventional Web Server implements some essential features of a conventional Web server. It consists of two sub-modules, shown in Figure 7.3 (b). In case the type of request is *normal*, the Response Composer sub-module of Conventional Web Server constructs the response, whereas in case the type of request is *MXML-request*, the response is produced in MXML Response Composer and is then passed to the *Response Sender* sub-module of Conventional Web Server.

- *Response Composer*: The Response Composer analyzes *normal* requests, and constructs responses sent to the client. As it is depicted in Figure 7.3 (b), this sub-module is engaged only if the request is of type *normal*.
- *Response Sender*: The Response Sender is responsible for sending the response to the client. As stated above, the response can originate either from the Response Composer, or from the MXML Response Composer.

7.5.3 System Implementation

The system components described in the previous section are in fact implemented as two separate programs. The MXML Web Server, except View Extractor, was developed in Java [JAVA, CL97]. The View Extractor has been implemented in C [KR88].

When an MXML document is requested (like, for example, the document `menu.mxml`), the MXML Web Server determines the dimensions that are involved and their respective domains by checking the corresponding MDTD, if it exists, or by scanning the MXML document itself, and by adding any extra dimensions that appear in the MXSL document. Then, the MXML Web Server presents the user with drop-down lists for assigning values to dimensions, as depicted in the upper left *dimension frame* of Figure 7.4.

Figure 7.4: A screenshot showing a reply from *MXML Web Server*.

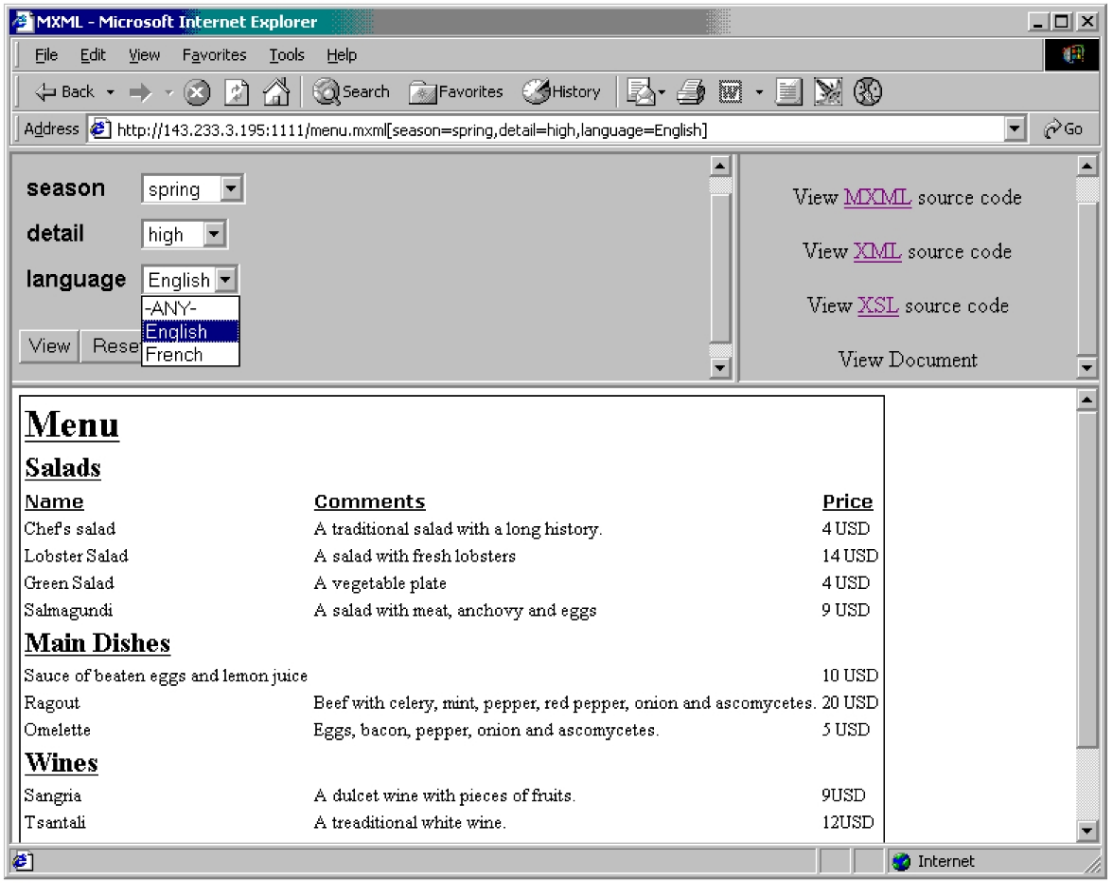


Figure 7.4 is a screenshot of the MXML Web Server replying to the request:

```
http://143.233.3.195:1111/menu.mxml[season=spring,
                                    detail=high,language=English]
```

The context specifier in this extended URL is composed dynamically by JavaScript [JSCR] code, based on the values selected by the user for the dimension drop-down lists in the upper left *dimension frame*. Notice that for every dimension there exists a value "ANY" in the corresponding drop-down list. The meaning of "ANY" is that no value is specified for that

dimension. Consequently, if a context specifier contains one or more "ANY", it specifies more than one world. If it does not contain "ANY" at all, it specifies exactly one world.

The upper-right *action frame* allows the user to view the MXML source code, and the MXSL source code. If the selected context specifies exactly one world, then the user may also see the XML and XSL sources of the corresponding facets under that world, plus the final output document. The source code and the final document are displayed in the lower *output frame*, which in Figure 7.4 shows the multidimensional menu of a restaurant under the world `{(season,spring), (detail,high), (language,English)}`. If the context specified by the user represents more than one world, then the *output frame* displays the MXML and MXSL produced by partial reduction for that context[116]. In this case, the output frame displays a fixed message instead of the final document, stating that for the final document to appear the user must give a value other than "ANY" to every dimension.

7.6 SUMMARY

In this chapter we proposed **Multidimensional XML**, or **MXML** in short, an extension of XML suitable for representing data that assume facets with different value or different structure under different contexts. In MXML, elements and attributes may depend on a number of dimensions, which define the worlds under which the facets of those elements or attributes hold. Moreover, we proposed an extension of DTD, called **Multidimensional DTD** or **MDTD** in short, that describes the logical structure of MXML documents. We discussed the properties of MXML and its relation to Multidimensional Data Graph and MOEM, and gave examples of how an MXML document can be reduced to a conventional XML document under a given world.

In addition, we presented a new paradigm for handling multifaceted, context-dependent Web data. The new paradigm is called **multidimensional paradigm**, and comprises representation, manipulation and presentation issues. We discussed the presentation of MXML documents through **multidimensional XSL** (**MXSL**) stylesheets. We described the design of a system that implements the basic functionality of the multidimensional paradigm, and demonstrates how a user can interact with a multidimensional document and view different variants under different worlds.

[116] It is not possible for the user to specify *any* possible context. For example, the context `[season in {spring,winter} | season=summer,detail=low]` cannot be formulated using the *dimension frame* of Figure 7.4. The main objective of the implemented system is reduction to XML under a specific world, and partial reduction for a set of worlds is supported only marginally.

8 A SYSTEM FOR MANAGING MSSD

In this chapter, we present a system that implements essential concepts from previous chapters, and supports the management of multidimensional semistructured data. In particular, through the MOEM basic change operations defined in Chapter 6, the system allows the development and maintenance of Multidimensional Data Graphs, which are displayed in a graphical interactive environment. It implements context operations as introduced in Chapter 3, and calculates the inherited context of graphs, defined in Chapter 4. It performs reduction to OEM, partial reduction, and transformation to canonical form; the results of those operations are displayed as new graphs in the same interactive environment, and can be further manipulated just like the original graphs. The system can load and save MOEMs in a variety of formats, including mssd-expressions and MXML. Moreover, it encompasses a query subsystem that evaluates MQL queries, as described in Chapter 5.

The system is implemented in Java [JAVA, CL97], and can play the role of an infrastructure for developing new applications and MSSD tools that need to use the supported functionality. Such an application is *OEM History*, presented in Chapter 6, which relies on the system to perform lower level operations on MSSD. *MSSDesigner* is the graphical user interface for handling and querying Multidimensional Data Graphs, and provides an interactive way of using the system.

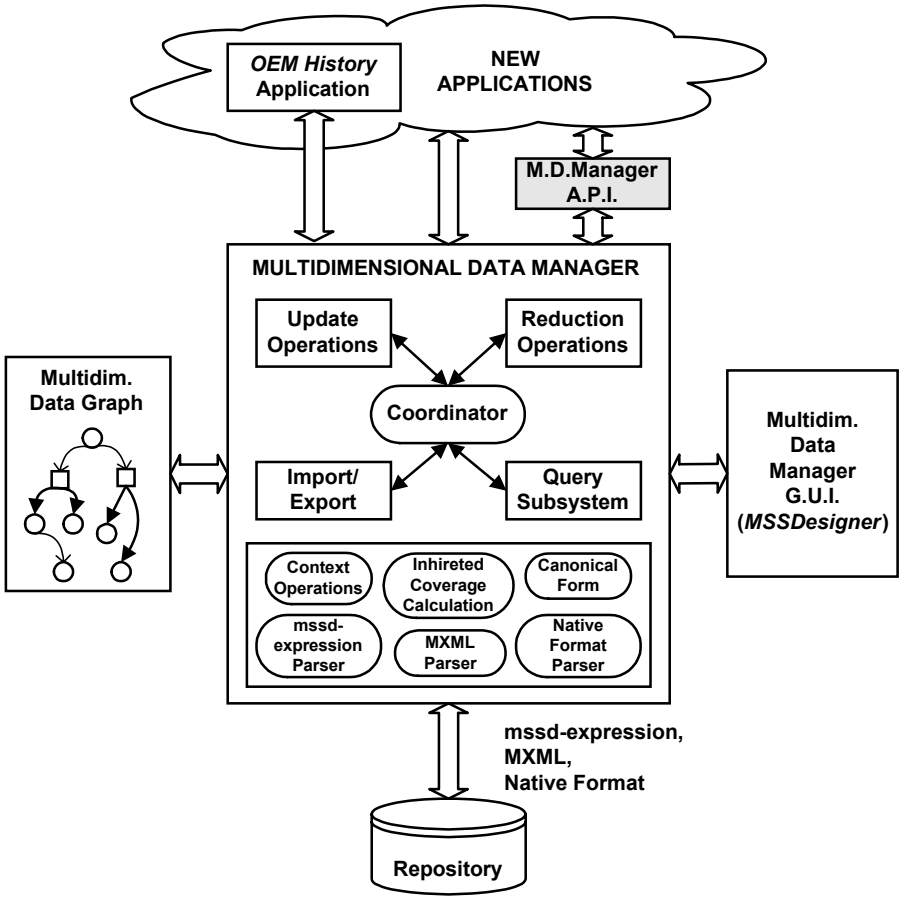
In what follows, we give an overview of the system, and then discuss its basic components one by one.

8.1 SYSTEM ARCHITECTURE

The implemented system is a prototype MSSD infrastructure that comprises a set of tools and processes for creating, manipulating, and querying Multidimensional Data Graphs. The system can be used directly through a graphical user interface, or it can provide support to applications that need an MSSD framework.

The overall architecture of the system is shown in Figure 8.1. We now discuss briefly its various components.

Figure 8.1: Overall architecture of the system.



Multidimensional Data Graph: This component consists of the main memory data structures that hold graph representations of MSSD.

Repository: It is the physical storage medium for loading and saving MSSD from and to files. A number of formats are supported, namely mssd-expressions, MXML, and Native Format.

Multidimensional Data Manager, or *MDM* in short: MDM is responsible for managing Multidimensional Data Graphs and MOEM graphs. It comprises a set of modules that support the creation, maintenance, and querying of multidimensional semistructured data. MDM converts the graph representation to other formats in order to save MSSD to files, and loads graphs from files in various MSSD formats. MDM is discussed further in Section 8.2.

MSSDesigner: MSSDesigner is a graphical user interface for MDM. It provides easy, interactive access to the various functions of MDM for graph creation, maintenance, and querying. MSSDesigner is presented in more detail in Section 8.3.

MDM Application Programming Interface: The MDM API is a library of commands that gives access to the functionality of the system. New applications will be able to use the existing MSSD infrastructure by issuing commands in the form of a script language. This component has not been implemented yet.

Applications: New applications can be built on top of the system, using its functionality through the MDM API. In addition, applications can use the system directly, as in the case of *OEM History* that is described in detail in Chapter 6.

MDM and MSSDesigner are obviously the most important of the components, and they are the topic of the following sections.

8.2 MULTIDIMENSIONAL DATA MANAGER

The modules included in MDM are shown in Figure 8.1. We start with a number of utility functions, which appear in a box at the bottom of MDM in Figure 8.1, and are directly accessible to all other MDM modules.

8.2.1 General Utility Functions

Context Operations implements operations on context specifiers defined in Chapter 3, including context intersection, context union, and simplification of context specifiers.

Inherited Coverage Calculator is used to compute inherited coverages in Multidimensional Data Graphs. At the moment, only the inherited context is calculated, however the same techniques can be used for calculating context coverage and inherited coverage, as explained in Chapter 4. For calculating the inherited context, the graph is traversed in a breadth-first manner, starting from the root. A simple fixed-point algorithm is implemented to deal with cycles in the graph.

Canonical Form transforms a graph to its canonical form.

The rest of the processes parse expressions in various formats, and are used to save a graph to a file, or to load a graph from a file. Those are *mssd-expression Parser*, *MXML Parser*, and *Native Format Parser*, which parse mssd-expressions, MXML, and "Native Format" expressions, respectively.

8.2.2 Coordinator Module

External components communicate with MDM through the *Coordinator* module. Its job is to decompose incoming requests into a number of basic operations, and to assign these operations to appropriate modules inside MDM.

As an example, consider the case where a user removes an edge from a graph through a graphical interface. The Coordinator accepts the request from MDM GUI, and diverts it to the *Update Operations* module, which is responsible for carrying out such modifications. After that, the Coordinator notifies the MDM GUI to redisplay the graph.

8.2.3 Update Operations Module

The *Update Operations* module carries out modifications to a Multidimensional Data Graph. It implements the basic change operations that were introduced in Chapter 6, namely *createCNode*, *updateCNode*, *createMNode*, *addEEdge*, *remEEdge*, *addCEdge*, and *remCEdge*.

As in conventional OEM, in Multidimensional Data Graph object deletion is achieved through edge removal, since the persistence of an object is determined by whether the object

is reachable from the root of the graph. The module can be requested to check the graph for inaccessible nodes, which are then automatically removed.

8.2.4 Import / Export Module

A Multidimensional Data Graph can be encoded and stored in a file in a number of different formats. Currently, the system supports three different formats: mssd-expressions that are defined in Chapter 4, Multidimensional XML that is defined in Chapter 7, and Native Format expressions. The *Import / Export* module handles the storing and the loading process of a graph in those formats.

In contrast to other formats, Native Format expressions retain the position on the screen of nodes and edges in a graph. Every node is described with a line of the form

```
[nodeId, nodeType, xPos, yPos, value, isRoot, bHung]
```

where `nodeId` is the oid of the node, `nodeType` is the type of the node (atomic, complex, or multidimensional), `xPos` and `yPos` are the node coordinates on the screen canvas, and `value` is the value of the node. In addition, `isRoot` is true if the node is the root of the graph, and `bHung` is true if the node is not reachable from the root.

Every edge is described with a line of the form

```
(fromNode; toNode; edgeType; value)
```

where `fromNode` is the node of departure, `toNode` is the destination node, `edgeType` specifies whether the edge is a context edge or an entity edge, and `value` is the edge label: a context specifier (explicit context) if the edge is a context edge, and a string label if the edge is an entity edge.

8.2.5 Reduction Operations Module

This module implements reduction to an OEM holding under a specific world, and partial reduction for a given context. Those processes are both defined in Chapter 4. The system currently uses inherited context to implement reduction, instead of inherited coverage as described in Chapter 4. The reason is that, as stated in Section 8.2.1, the system for the moment does not calculate context coverage. As a consequence, it is possible that in some cases reduction to OEM will return a graph with complex nodes as leaves.

8.2.6 Query Subsystem

The *Query Subsystem* implements MQL on top of LORE [MAG+97], as described in detail in Chapter 5. It is responsible for executing MQL queries on a Multidimensional Data Graph, and for producing new Multidimensional Data Graphs that are the results of those queries. As mentioned in Section 8.2.1, our system does not currently calculate context coverage. For this reason, when evaluating MQL queries the inherited context of graphs is used instead of the inherited coverage.

Our system is implemented in Java, and runs on any operating system supported by Java. On the other hand, LORE runs on LINUX [LIN], which puts a constraint on our system if we want the Query Subsystem to be operational. To solve this, we implemented a server program in Java that acts as an intermediary between the Query Subsystem and LORE, and exchanges

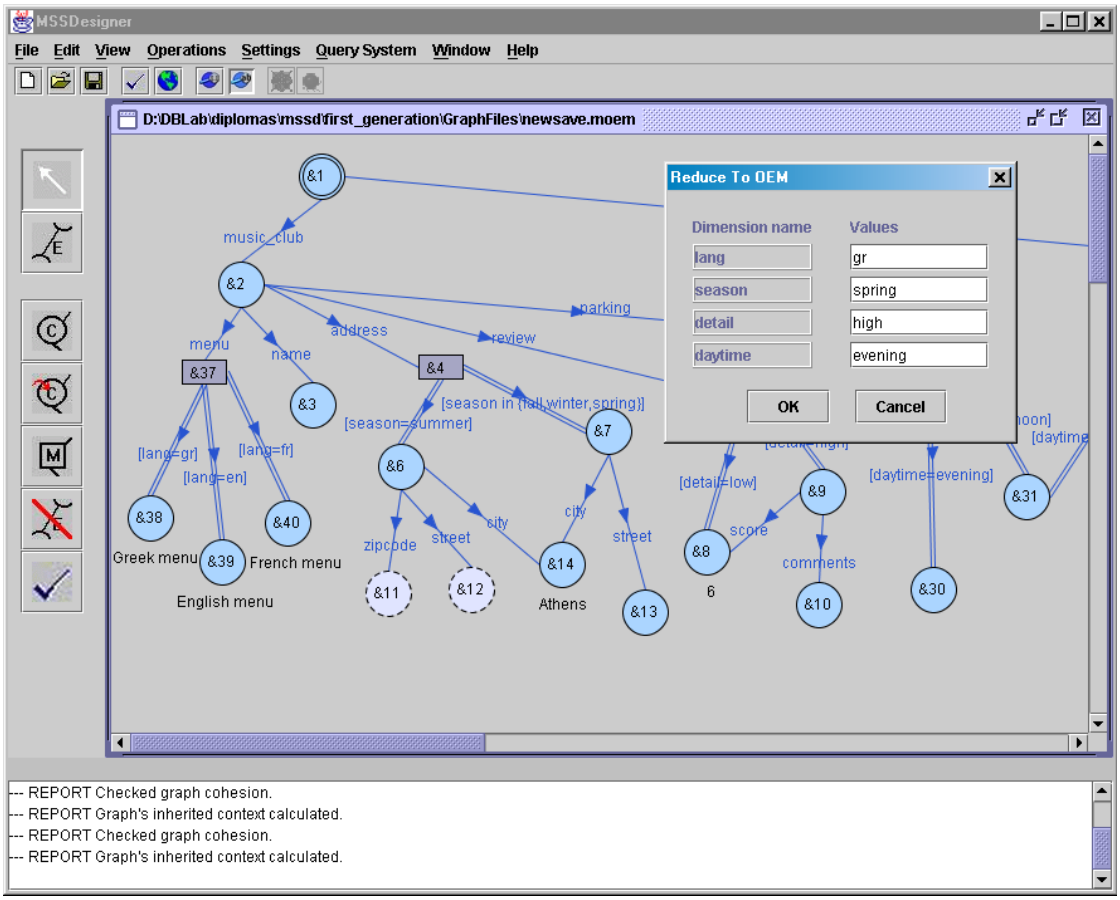
data with both systems using a special protocol. This server program operates on the same LINUX machine as LORE, and allows our system to use LORE from another machine.

The following section describes MSSDesigner, and also gives a concrete view of how the Query Subsystem can be used.

8.3 MSSDESIGNER

MSSDesigner is a graphical user interface that gives access to the functionality of Multidimensional Data Manager. Figure 8.2 shows a sample screenshot of MSSDesigner, displaying a graph for the multidimensional recreation guide example that was presented in previous chapters. On the right side a dialog box prompts the user to specify a world, and reduce the graph to OEM.

Figure 8.2: A sample screenshot of *MSSDesigner*.



MSSDesigner employs a multi-document interface (MDI), where many windows can be open simultaneously, each displaying a different graph. All operations are performed through the menu items and the toolbar buttons, and have effect on the graph whose window has currently the focus. The buttons in MSSDesigner are positioned on two toolbars, placed at the upper and left sides of the main window. The toolbar on the left side contains buttons

corresponding to the basic change operations that modify the graph. Starting from the top of the toolbar, the arrow-labeled button allows multiple selection and transposition of nodes in the display area. The following five buttons correspond to operations for adding and updating nodes, and for adding and removing edges. The last button performs consistency check of the graph, and removes nodes that are not accessible from the root.

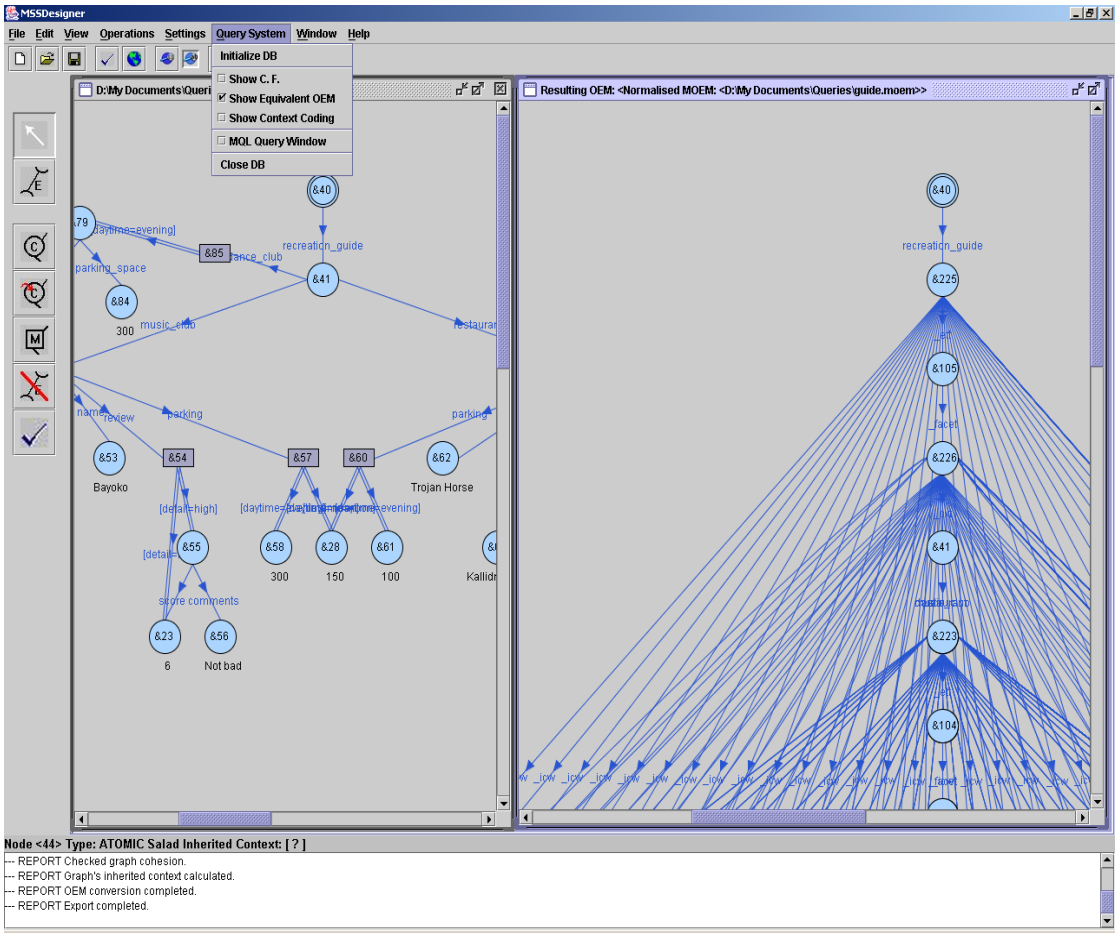
The upper toolbar contains nine buttons. The first one opens an empty window for designing a new graph. The second button creates a graph by importing Native Format expressions, whereas the next button exports a graph in Native Format. The button with the tick symbol checks the validity of the graph[117], while the next button is used to reduce the graph to an OEM holding under a specified world. Partial reduction can be performed through the application menu. It is possible to display or hide the explicit contexts and labels of edges with the following button. The next button calls the Inherited Coverage Calculator, and causes inherited contexts[118] to appear on the screen. The last couple of buttons open the *MQL Query* window, and a window showing the integers used to encode worlds (as explained in Chapter 5). All button operations can also be performed via the MSSDesigner menu.

With MSSDesigner we can create a graph from an mssd-expression or an MXML document, and export an existing graph to those formats. The File Import / Export menu items correspond to those operations. Native Format expressions store the position of nodes on the screen as well, and are useful for saving and loading unfinished graphs, because the process of saving and loading in Native Format does not perform consistency checking, thus hanging nodes / subgraphs are not removed. On the other hand, the graph is always checked for consistency when importing or exporting mssd-expressions or MXML documents.

Figure 8.3 shows a screenshot of MSSDesigner after having initialized the Query Subsystem. The initialization of the Query Subsystem performs the following steps: (1) the graph G in the window with the focus is transformed to a new graph G_{CF} that is in canonical form, (2) possible worlds for G are encoded as integers, (3) the graph G_{CF} is transformed to an equivalent OEM O through the process **MDGToOEM** defined in Chapter 5, and (4) the resulting OEM O is passed on to LORE and becomes the current database.

[117] The notion of **validity** is defined in [SG02], however *context coverage* is a more powerful concept, and its introduction made validity obsolete.
[118] For the moment, inherited contexts are calculated and displayed instead of inherited coverages.

Figure 8.3: Initializing the Query Subsystem with a graph database.

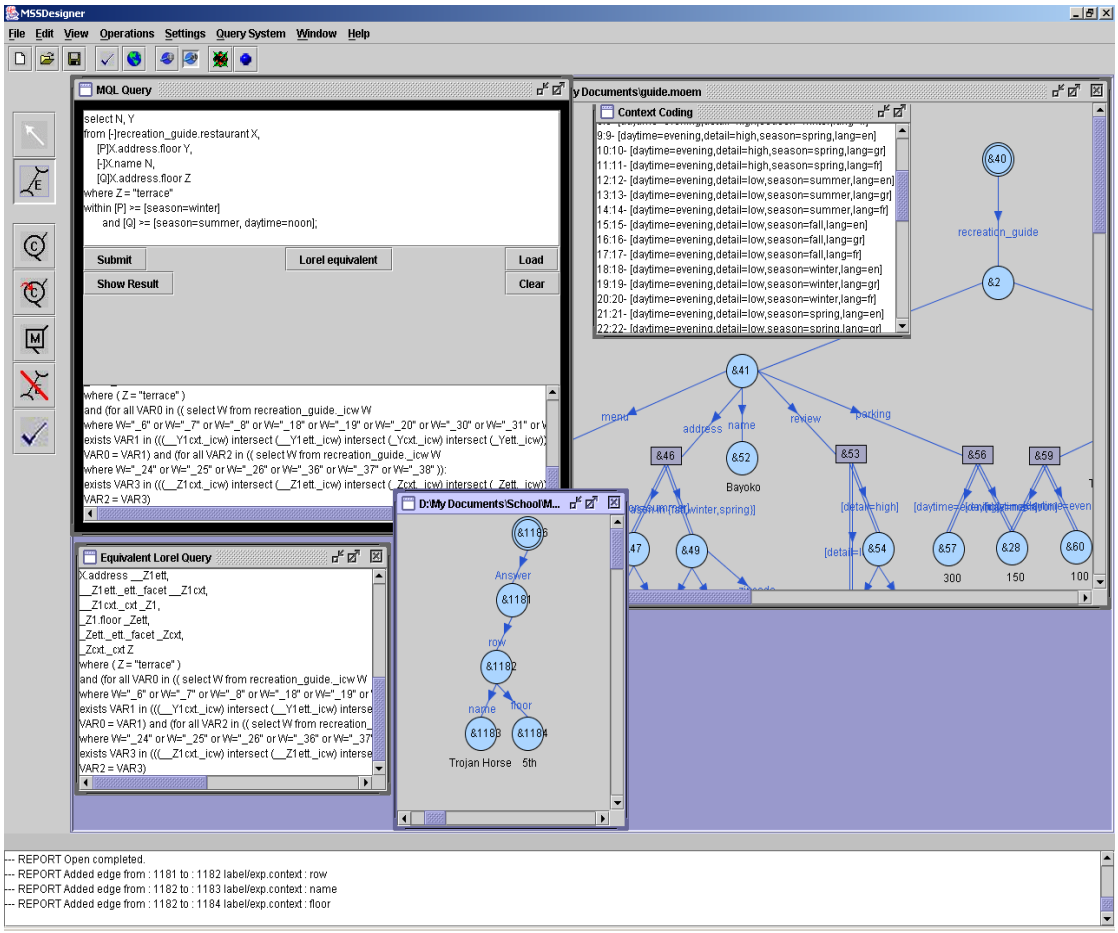


As shown in Figure 8.3, after selecting "Initialize DB" from the menu, the user can open a new window with the graph database in canonical form, or with the equivalent OEM graph that is passed on to LORE. The right window of Figure 8.3 displays the equivalent OEM for the graph database that appears in the left window. In addition, the user can open a dialog box that lists the integers used to encode worlds.

Once the Query Subsystem has been initialized, the MQL Query window can be displayed. In Figure 8.4, a thick black border identifies the MQL Query window. The MQL query is inserted in the upper part of this window, either by typing or by loading text from a file. Then by pressing the button "Lorel equivalent", the equivalent[119] Lorel query appears in the lower part of the MQL Query window. As the equivalent Lorel query can span several lines, it is possible to display it in a separate window with caption "Equivalent Lorel Query", which is shown in the lower left part of Figure 8.4.

[119] As defined in Chapter 5.

Figure 8.4: Evaluating an MQL query and displaying the results.



The "Submit" button in the MQL Query window submits the equivalent Lorel query to LORE, and gets back the result as an XML document. By pressing the "Show Result" button, the following steps are performed: (1) the XML document with the result is transformed to an OEM graph, (2) the OEM graph is transformed to a Multidimensional Data Graph through the process **OEMToMDG** defined in Chapter 5, and (3) the Multidimensional Data Graph is displayed in a new window, as the final result of the MQL query.

In Figure 8.4, the small window with the graph in the lower part displays the result of the MQL query[120] that appears at the upper left part. Although the graph has the form of a conventional OEM, it is actually a Multidimensional Data Graph that happens to have only context nodes and entity edges. The integers used to encode worlds are converted back to context specifiers in the results. The small window in the upper right part of Figure 8.4 displays a list of integers together with the worlds they encode.

MQL queries can be submitted one after another to the same MOEM database; however, in order to change the database to another graph, the Query Subsystem must be re-initialized through the "Initialize DB" menu item. The "Close DB" menu item can be used to terminate

[120] A careful look may reveal some minor differences between this MQL query and the MQL syntax as specified in Chapter 5. The reason is that the Query Subsystem implements the first version of MQL, which has some syntactic differences with the current version, defined in Chapter 5.

the connection with LORE, and to perform cleanup operations in Multidimensional Data Manager and in LORE.

8.4 SUMMARY

In this chapter we presented a system for managing MSSD that implements most of the essential concepts introduced in previous chapters, and can be used as an infrastructure for the development of new MSSD tools and applications. We discussed the architecture of the system, its modules, and its operation through *MSSDesigner*, a graphical user interface that supports the interactive creation, experimentation, and querying of Multidimensional Data Graphs through MQL.

9 CONCLUSIONS AND FUTURE WORK

Contrary to traditional databases and information systems where the number of users is more or less known and their background is to a great extent homogeneous, Web users do not share the same background and do not apply the same conventions when interpreting data. Such users can have different perspectives of the same information entities, a situation that should be taken into account by Web data models and query languages. Moreover, information providers often need to manage different variations of essentially the same data, which are targeted to different consumer groups. In this thesis we propose **multidimensional semistructure data** (**MSSD**), where information entities may manifest different facets, according to *context*. What follows summarizes our approach and the contribution of this thesis.

We developed a formalism for representing context in a flexible and intuitive way, using variables called **dimensions**. **Context specifiers** define contexts by constraining the possible values dimensions can take, and by combining such dimension constraints in conjunctions and disjunctions. We gave semantics to context by showing that it can be viewed as a set of possible **worlds**, where a world is an environment under which information obtains an unambiguous interpretation. We defined a number of context operations for combining and comparing contexts, and examined their properties and time complexities. An important advantage of our approach is that most context operations can be performed even if our knowledge of the dimensions involved is incomplete; in this case the correspondence of contexts to sets of worlds can be deferred until knowledge of dimensions is complete.

We proposed **Multidimensional Data Graph**, a graph data model for MSSD that incorporates context specifiers. In Multidimensional Data Graph, labeled edges define relationships between multidimensional entities, which are information entities that may manifest different facets under different worlds. Context specifiers are used to qualify those facets, and define constraints on the worlds under which a facet may hold. In order to get the actual worlds under which graph elements really hold, we must take into account the accumulated constraints as they propagate from the root to the leaves and from the leaves to the root. This propagation of context ensures that a child node holds only under the worlds some parent node holds, and that a node holds only under the worlds it has access to some atomic node. Based on those principles, we defined how the *true* context of a node or edge is calculated in the frame of a graph. This context is called **inherited coverage**, while **path inherited coverage** represents the worlds under which a complete path holds.

We showed that conventional OEMs holding under some world can be extracted from a Multidimensional Data Graph, and we gave the conditions under which such extractions can take place. As a part of those conditions we discussed **context-deterministic** graphs, and we presented a process that **reduces to OEM** a context-deterministic Multidimensional Data Graph under a given world. In addition, we described **partial reduction** of a Multidimensional Data Graph for a given context. A process that transforms a Multidimensional Data Graph to **canonical form** has also been defined. Redundant dependencies between multidimensional entities may cause anomalies when maintaining a Multidimensional Data Graph, and the canonical form avoids those anomalies by

transforming the initial graph into a better-structured graph. An important property of a graph in canonical form is that every path has a regular structure, which is very useful when formulating queries. Having investigated the properties of Multidimensional Data Graph in depth, we introduced **Multidimensional OEM** (**MOEM**) as a special case of Multidimensional Data Graph that is a strict conglomeration of conventional OEMs holding under various worlds. We also specified **mssd-expression**, a syntax for expressing textually Multidimensional Data Graphs and MOEMs.

We specified **Multidimensional Query Language** (**MQL**), a query language for multidimensional semistructured data. MQL supports context and multifaceted entities as first class citizens, and can express *context-driven queries*. MQL is largely based on Lorel and retains its major contributions, namely path expressions and coercion. We defined **context path expressions**, which extend path expressions by incorporating **context qualifiers**. Context qualifiers can be context specifiers, **context variables**, and **context patterns**, and are used to pose context conditions on the database graph. We showed that a class of context path expressions can be viewed as defining joins between conventional OEMs that hold under different worlds. MQL uses two special clauses for handling context: the clause `within` expresses conditions on context, and the clause `context` creates new context variables to be used in the construction of results. In addition, the keyword `holding` can be used to apply partial reduction to the result graph. Our prototype implementation of MQL demonstrated how an MQL query compares to an equivalent Lorel query: in context-driven queries, MQL and MOEM are obviously far more expressive and elegant than Lorel and OEM, while in conventional queries MQL and context path expressions become as simple to formulate as Lorel and conventional path expressions. In our view, the potential of MSSD and the expressiveness of MQL justify the extensions to OEM we introduced in Multidimensional Data Graph. The additional node and edge types of Multidimensional Data Graph are used by MQL to formulate *cross-world queries*, which have no counterpart in context-unaware databases.

To assess our approach, we applied the MSSD framework we have developed to a classical problem in databases: we used MOEM and MQL to represent and query histories of OEM databases. We explained in detail how MOEM can be used to represent the history of an OEM database. We introduced the **basic change operations** of MOEM, and specified in pseudocode the process that encodes in an MOEM the history of an OEM database. We discussed how Multidimensional Data Graph properties apply in this particular case, and showed that temporal OEM snapshots can be obtained from MOEM. We presented *OEM History*, an implemented system, and demonstrated through an example the process of using an underlying MOEM database to model changes in OEM. In addition, we showed that MOEM is capable to model changes occurring not only in conventional OEM databases, but in MOEM databases as well. An important benefit of our approach stems from the queries that can be posed on OEM histories and MOEM histories. Our approach maintains snapshots of the successive states of an object, as facets of a multidimensional entity. This allows the formulation of queries that relate different states of the same or different object(s) at various time instances. We presented a number of MQL query examples, and through them we confirmed the expressiveness of MQL and the value of some of its features, like for example context patterns.

As an example of how the concepts we introduced can span other directions, we discussed **Multidimensional XML** (**MXML**), an extension of XML suitable for representing data that assume different value or different structure under different contexts. In MXML, elements and attributes may depend on a number of dimensions, which define the worlds under which the facets of those elements or attributes hold. Moreover, we proposed an extension of DTD, called **Multidimensional DTD** (**MDTD**), that describes the logical structure of MXML

documents. We discussed the properties of MXML, and gave examples of how an MXML document can be reduced to a conventional XML document under a given world. Moreover, we presented a new paradigm for handling multifaceted, context-dependent Web data. The new paradigm is called **multidimensional paradigm**, and comprises representation, manipulation, and presentation issues. We discussed the presentation of MXML documents through **multidimensional XSL** (**MXSL**) stylesheets. We described the design of a system that implements the basic functionality of the multidimensional paradigm, and demonstrates how a user can interact with a multidimensional document and view different variants under different worlds.

Finally, we presented a system for managing MSSD that implements most of the essential concepts introduced in this thesis, and can be used as an infrastructure for the development of new MSSD tools and applications. We discussed the architecture of the system, its modules, and its operation through *MSSDesigner*, a graphical user interface that supports the interactive creation, experimentation, and querying of Multidimensional Data Graphs through MQL.

This work opens a number of directions for further research, which we describe briefly in what follows.

- *Context representation*: We assumed that dimensions are orthogonal, and that their domains are flat sets enumerating elements. However, not all real-world applications comply with those assumptions. It would be interesting to support dimensions that are not orthogonal, or whose domains have a hierarchical structure, and investigate how this affects the notion of context and context operations.
- *Context-aware data models*: Efficient algorithms for calculating the inherited coverages of a Multidimensional Data Graph are an open issue. In addition, schemata for MOEM databases that include dimensions would be very useful for formulating queries. Moreover, context-nondeterministic graphs should be examined in more detail. Another interesting issue is methods that use context to visualize data models and to facilitate the update and maintenance of information. More important, concepts and properties from Multidimensional Data Graph can be applied to other types of data models, in order to develop object or relational databases that support context as first-class citizen.
- *MQL*: Evaluation of MQL queries involves a number of open research directions, like storage of Multidimensional Data Graphs, access methods, and optimization of context-driven queries. In addition, MQL could be extended to handle updates of Multidimensional Data Graphs.
- *Representing histories of OEM databases*: When using MOEM to represent database histories we assumed that time is linear. Supporting branching time is an interesting direction that would be useful in a number of real-world applications. In this case, changes may occur at any temporal snapshot of the database, not only at the current snapshot. The support of hierarchical dimension domains could be a first step towards this direction.
- *MXML*: XML-related technologies are currently a very active research area. A more rigorous investigation of how context can be introduced in those technologies is needed, including query languages for XML. Moreover, it would be interesting to adapt to XML our approach for representing histories of OEM databases, so as to be able to represent histories of XML data.

The role of context in managing the multiple aspects of Web data is attracting increasing attention. We believe that MSSD address an important problem of databases in today's global

environment, and that they have the potential to be practically applied in various domains. Thus, another interesting direction is to use MSSD concepts in problems from disperse fields, like:

- In providing personalized information and services, by adapting them to the profiles of consumers.
- In information integration, for modeling objects whose value or structure vary according to sources.
- In digital libraries, for representing metadata that conform to similar but not identical formats, depending on the source.
- In representing geographical information, where possible dimensions could be *scale* and *theme*.

APPENDIX A: MQL SYNTAX SPECIFICATION

In what follows, we specify the syntax of context path expressions and MQL in Extended Backus-Naur Form [EBNF], or EBNF for short.

Symbols that start with a capital letter can be defined by a regular expression, as for example in:

```
LabelName ::= [A-Za-z0-9_]+
```

Such symbols are not analyzed further here. Those include `Identifier`, which denotes strings indentifying variables, `CxtOid` and `MldOid`, which indicate oids of context and multidimensional objects respectively, `LabelName`, which denotes string labels of entity edges, and `QuotedStringLiteral`, `IntLiteral`, `RealLiteral`, which are self-explanatory.

Symbols that start with a lowercase letter are defined by an EBNF production, included below. Not included are the productions for `labelNamePattern` and `stringPattern`, which represent regular expressions on entity edge labels and strings respectively. Finally, the EBNF productions for `cxtSpec` and `dimList` are given in Chapter 3.

A.1 SYNTAX OF CONTEXT PATH EXPRESSIONS

Context path expressions are defined here in a bottom-up manner.

Table A.1 gives the productions for the variable types in MQL.

Table A.1: Variables.

```
cxtObjVar ::=  Identifier
mldObjVar ::=  "<" Identifier ">"
cxtVar    ::=  "[" Identifier "]"
labelVar  ::=  "$" Identifier
pathVar   ::=  "@" Identifier
```

Table A.2 defines the syntax of context patterns and context qualifiers.

Table A.2: Context pattern and context qualifier.

```
cxtPattern   ::=  "[%" dimList ("|" dimList)* "]"
cxtQualifier ::=  cxtSpec | cxtPattern | cxtVar
```

Table A.3 defines the syntax of entity parts and facet parts. Entity parts that may be followed by a label variable, and facet parts that may include a context variable as explicit

context qualifier are named **augmented parts**. A root part is defined as a special entity part that does not begin with a dot.

Table A.3: Entity part and facet part.

```
ettLabelExpr     ::=  "–" | LabelName | "~" labelNamePattern "/"
ettPart          ::=  "." cxtQualifier? ettLabelExpr
ettPartAugment   ::=  ettPart labelVar?
facetPart        ::=  "::" cxtQualifier? (cxtSpec | cxtPattern)
facetPartAugment ::=  "::" cxtQualifier? cxtQualifier
rootPart         ::=  cxtQualifier? ettLabelExpr
rootPartAugment  ::=  rootPart labelVar?
                      | rootPartAugment "{" cxtObjVar "}"
                      | rootPartAugment "{" mldObjVar "}"
```

Table A.4 gives the syntax rules for gcpe_components. Four types of components are defined, depending on the types of the first and the final parts. The productions do not allow a facet part to follow another facet part.

Table A.4: General context path expression components.

```
gcpeComponEtt-Ett ::= ettPart
         | gcpeComponEtt-Ett gcpeComponEtt-Ett
         | gcpeComponEtt-Ett gcpeComponFacet-Ett
         | gcpeComponEtt-Facet gcpeComponEtt-Ett
         | "(" gcpeComponEtt-Ett "|" gcpeComponEtt-Ett ")"
         | "(" gcpeComponEtt-Ett ")" "+"
gcpeComponEtt-Facet ::= "." cxtQualifier? "#"
         |  gcpeComponEtt-Ett gcpeComponEtt-Facet
         |  gcpeComponEtt-Ett gcpeComponFacet-Facet
         |  gcpeComponEtt-Facet gcpeComponEtt-Facet
         | "(" gcpeComponEtt-Facet "|" gcpeComponEtt-Facet ")"
         | "(" gcpeComponEtt-Facet ")" ("*" | "?" | "+")
gcpeComponFacet-Facet ::= facetPart
         | gcpeComponFacet-Ett gcpeComponEtt-Facet
         | gcpeComponFacet-Ett gcpeComponFacet-Facet
         | gcpeComponFacet-Facet gcpeComponEtt-Facet
         | "(" gcpeComponFacet-Facet "|" gcpeComponFacet-Facet ")"
         | "(" gcpeComponFacet-Facet ")" "?"
gcpeComponFacet-Ett ::= "::" cxtQualifier? "#"
         |  gcpeComponFacet-Ett gcpeComponEtt-Ett
         |  gcpeComponFacet-Ett gcpeComponFacet-Ett
         |  gcpeComponFacet-Facet gcpeComponEtt-Ett
         | "(" gcpeComponFacet-Ett "|" gcpeComponFacet-Ett ")"
         | "(" gcpeComponFacet-Ett ")" ("*" | "?" | "+")
```

In Table A.5, gcpe_components are augmented with label, path, and object variables, which are not allowed inside gcpe_components. Moreover, in an **augmented gcpe_component** a context variable can be used as an explicit context qualifier (a sort of label variable for context edges), which again is not allowed inside a gcpe_component[121].

[121] Augmented gcpe_components correspond to *qualified gcpe_components* of Lorel [AQM+97].

Table A.5: Augmented general context path expression components.

```
gcpeAugmentEtt-Ett ::= ettPartAugment
                     | gcpeComponEtt-Ett
                     | gcpeAugmentEtt-Ett gcpeAugmentEtt-Ett
                     | gcpeAugmentEtt-Ett gcpeAugmentFacet-Ett
                     | gcpeAugmentEtt-Facet gcpeAugmentEtt-Ett
                     | "(" gcpeAugmentEtt-Ett ")" pathVar
                     | gcpeAugmentEtt-Ett "{" cxtObjVar "}"
                     | gcpeAugmentEtt-Ett "{" mldObjVar "}"
gcpeAugmentEtt-Facet ::=
                       gcpeComponEtt-Facet
                     | gcpeAugmentEtt-Ett gcpeAugmentEtt-Facet
                     | gcpeAugmentEtt-Ett gcpeAugmentFacet-Facet
                     | gcpeAugmentEtt-Facet gcpeAugmentEtt-Facet
                     | "(" gcpeAugmentEtt-Facet ")" pathVar
                     | gcpeAugmentEtt-Facet "{" cxtObjVar "}"
gcpeAugmentFacet-Facet ::= facetPartAugment
                     | gcpeComponFacet-Facet
                     | gcpeAugmentFacet-Ett gcpeAugmentEtt-Facet
                     | gcpeAugmentFacet-Ett gcpeAugmentFacet-Facet
                     | gcpeAugmentFacet-Facet gcpeAugmentEtt-Facet
                     | "(" gcpeAugmentFacet-Facet ")" pathVar
                     | gcpeAugmentFacet-Facet "{" cxtObjVar "}"
gcpeAugmentFacet-Ett ::=
                       gcpeComponFacet-Ett
                     | gcpeAugmentFacet-Ett gcpeAugmentEtt-Ett
                     | gcpeAugmentFacet-Ett gcpeAugmentFacet-Ett
                     | gcpeAugmentFacet-Facet gcpeAugmentEtt-Ett
                     | "(" gcpeAugmentFacet-Ett ")" pathVar
                     | gcpeAugmentFacet-Ett "{" cxtObjVar "}"
                     | gcpeAugmentFacet-Ett "{" mldObjVar "}"
```

Table A.6 defines context path expressions as an identifier of some type, followed by an augmented gcpe_component of a compatible type (an augmented gcpe_component may comprise any number of augmented gcpe_components). Two types of context path expressions are distinguished, depending on the type of their final part.

Table A.6: Context path expressions.

```
cxtPathExpr-Ett ::=
   (CxtOid | cxtObjVar) gcpeAugmentEtt-Ett
 | (MldOid | mldObjVar | rootPartAugment) (gcpeAugmentFacet-Ett
                                          | gcpeAugmentEtt-Ett)

cxtPathExpr-Facet ::=
   (CxtOid | cxtObjVar) gcpeAugmentEtt-Facet
 | (MldOid | mldObjVar | rootPartAugment) (gcpeAugmentFacet-Facet
                                          | gcpeAugmentEtt-Facet)

cxtPathExpr  ::=  cxtPathExpr-Ett | cxtPathExpr-Facet
```

Having defined context path expressions, we continue with the syntax of MQL per se.

A.2 MQL SYNTAX

The syntax rules of MQL are listed in a top-down manner.

Table A.7 gives the overall structure of an MQL query. We distinguish between two types of queries, depending on the type of edges that depart from the root of the results, as specified in the `select` clause. Only queries of the same type can combine their results using `union`, `intersect`, and `except`.

Table A.7: Top level structure of MQL queries.

```
query    ::=  queryEtt | queryCxt

queryEtt ::=  clauseQueryEtt
              | queryEtt "union" queryEtt
              | queryEtt "intersect" queryEtt
              | queryEtt "except" queryEtt
              | "(" queryEtt ")"
queryCxt ::=  clauseQueryCxt
              | queryCxt "union" queryCxt
              | queryCxt "intersect" queryCxt
              | queryCxt "except" queryCxt
              | "(" queryCxt ")"

clauseQueryEtt ::=
        "select" "holding"? "distinct"?
        (labeledTemplateEtt | "{" templateEtt "}")
        queryBody
clauseQueryCxt ::=
        "select" "holding"? "distinct"?
        (labeledTemplateCxt | "<" templateCxt ">")
        queryBody

queryBody ::= ("context" contextExpr ("," contextExpr)*)?
              ("from" fromExpr ("," fromExpr)*)?
              ("where" predicate)?
              ("within" cxtPredicate)?
```

Table A.8 defines the syntax of the mssd-expression template in `select`. The first label in the template cannot be missing if the outer brackets that define the type of the root are missing.

Table A.8: Template of results in "select" clause.

```
labeledTemplateEtt ::=
        templObjLabelEtt ":" templObjValue ("," templateEtt)?
labeledTemplateCxt ::=
        templObjLabelCxt ":" templObjValue ("," templateCxt)?

templateEtt ::=
     (templObjLabelEtt ":")? templObjValue ("," templateEtt)?
templateCxt ::=
     (templObjLabelCxt ":")? templObjValue ("," templateCxt)?

templObjLabelEtt ::=  LabelName | labelVar
templObjLabelCxt ::=  cxtSpec   | cxtVar

templObjValue  ::=  "{" templateEtt "}" | "<" templateCxt ">"
                    | atomicValue | cxtPathExpr | safeQuery

safeQuery ::= "(" query ")"
```

Table A.9 defines atomic values that may take the place of objects in the template of `select`, or be used in the `where` clause for comparisons.

Table A.9: Object values in the template of "select" clause.

```
atomicValue  ::=  variable | constant | numValue
                | "oid" "(" (cxtObjVar | mldObjVar) ")"
                | "pathof" "(" pathVar ")"

variable ::= cxtObjVar | mldObjVar | cxtVar | labelVar | pathVar

constant     ::=  QuotedStringLiteral | numConstant
numConstant  ::=  IntLiteral | RealLiteral

numValue ::= cxtObjVar | numConstant
           | numValue arithOp numValue
           | "(" numValue ")"
           | "-" numValue
           | "abs" "(" numValue ")"
           | numAggrFunct "(" (cxtObjVar | cxtPathExpr-Facet) ")"
           | "count" "(" (variable | cxtPathExpr) ")"

arithOp      ::=  "+" | "-" | "*" | "/" | "mod"
numAggrFunct ::=  "min" | "max" | "sum" | "avg"
```

Table A.10 specifies the definition of a new context variable in the `context` clause.

Table A.10: Components in "context" clause.

```
contextExpr ::= cxtVar "as" (cxtValue
                             | "extension" "(" cxtValue ")")
cxtValue    ::= cxtSpec | cxtVar
              | cxtValue cxtOp cxtValue
              | cxtValue cxtOp cxtPattern
              | cxtPattern cxtOp cxtValue
              | "(" cxtValue ")"
              | "union" "(" cxtValue ")"
              | "intersect" "(" cxtValue ")"

cxtOp  ::=  "+" | "-" | "*"
```

Table A.11 gives the syntax of a `from` clause component, consisting of a context path expression and (maybe) an object variable of compatible type.

Table A.11: Components in "from" clause.

```
fromExpr ::= cxtPathExpr-Facet ("as"? cxtObjVar)?
             | cxtPathExpr-Ett ("as"? (cxtObjVar | mldObjVar))?
             | rootPartAugment ("as"? (cxtObjVar | mldObjVar))?
```

Table A.12 defines the syntax of the `where` clause.

Table A.12: Structure of "where" clause.

```
predicate ::= "not" predicate
   | predicate "and" predicate
   | predicate "or" predicate
   | "(" predicate ")"
   | boolConstant
   | (atomicValue | cxtPathExpr) "like" "“" stringPattern "”"
   | (atomicValue | cxtPathExpr) compOp compQuantifier?
                        (atomicValue | cxtPathExpr | safeQuery)
   | "exists" (variable | cxtPathExpr)
   | "exists" cxtObjVar "in" cxtPathExpr-Facet "(" predicate ")"
   | "exists" (cxtObjVar | mldObjVar) "in"
              (cxtPathExpr-Ett | safeQuery) "(" predicate ")"
   | "for all" cxtObjVar "in" cxtPathExpr-Facet "(" predicate ")"
   | "for all" (cxtObjVar | mldObjVar) "in"
               (cxtPathExpr-Ett | safeQuery) "(" predicate ")"

boolConstant   ::=   "true" | "false"
compOp         ::=   "<" | "<=" | "=" | "==" |"!=" | ">=" | ">"
compQuantifier ::=   "any" | "all"
```

Table A.13 defines the syntax of the `within` clause.

Table A.13: Structure of "within" clause.

```
cxtPredicate ::= "not" cxtPredicate
                 | cxtPredicate "and" cxtPredicate
                 | cxtPredicate "or" cxtPredicate
                 | "(" cxtPredicate ")"
                 | boolConstant
                 | cxtValue cxtCompOp cxtValue
                 | cxtValue cxtCompOp cxtPattern
                 | cxtPattern cxtCompOp cxtValue

cxtCompOp  ::=  "<" | "<=" | "=" | "!=" | ">=" | ">"
```

This concludes the syntax specification of MQL.

REFERENCES

[Abi97] Serge Abiteboul. Querying Semi-Structured Data. In *Proceedings of the 6th International Conference on Database Theory (ICDT'97)*, Delphi, Greece, January 1997.

[Abi99] Serge Abiteboul. On Views and XML. In *Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'99)*, May 1999.

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.

[AFJ91] E. A. Ashcroft, A. A. Faustini, and R. Jagannathan. An Intensional Language for Parallel Applications Programming. *Parallel Functional Languages and Compilers*, pages 11-49. ACM Press, 1991.

[AFJW95] E. A. Ashcroft, A. A. Faustini, R. Jagannathan, and W. W. Wadge. *Multidimensional Programming*. Oxford University Press, 1995.

[AM98] Gustavo O. Arocena, and Alberto O. Mendelzon. WebOQL: Restructuring Documents, Databases and Webs. In *Proceedings of the International Conference on Data Engineering (ICDE'98)*, Orlando, Florida, 1998.

[AMM97] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. To Weave the Web. In *Proceedings of the 23rd International Conference on Very Large DataBases (VLDB'97)*, Athens, Greece, August 1997.

[AMM97a] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. Semistructured and Structured Data in the Web: Going Back and Forth. *SIGMOD Record*, 26(4):16-23, December 1997.

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Jannet L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1): 68-88, April 1997.

[AYU00] Toshiyuki Amagasa, Masatoshi Yoshikawa, and Shunsuke Uemura. A Data Model for Temporal XML Documents. In *Proceedings of the 11th International Conference on Database and Expert Systems Applications (DEXA'00)*, London, September 2000. Lecture Notes in Computer Science (LNCS) 1873, pages 334-344, Springer-Verlag, 2000.

[BBC+98] Phil Bernstein, Michael Brodie, Stefano Ceri, David DeWitt, Mike Franklin, Hector Garcia-Molina, Jim Gray, Jerry Held, Joe Hellerstein, H. V. Jagadish, Michael Lesk, Dave Maier, Jeff Naughton, Hamid Pirahesh, Mike Stonebraker, and Jeff Ullman. The Asilomar Report on Database Research. *SIGMOD Record*, 27(4): 74-80, 1998.

[BC00] Angela Bonifati, and Stefano Ceri. Comparative Analysis of Five XML Query Languages. *SIGMOD Record*, 29(1), March 2000.

[BCW93] M. Baudinet, J. Chomicki, and P. Wolper. Temporal Deductive Databases. *Temporal Databases: Theory, Design, and Implementation*, pages 294-320. The Benjamin / Cummings Publishing Company, 1993.

[BDHS96] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A Query Language and Optimization Techniques for Unstructured Data. In *Proceedings of the ACM International Conference on Management of Data*, 1996.

[BDS95] Peter Buneman, Susan Davidson, and Dan Suciu. Programming Constructs for Unstructured Data. In *Proceedings of the Workshop on Database Programming Languages*, Italy, 1995.

[BDT98] Peter Buneman, Alin Deutsch, and Wang-Chiew Tan. A Deterministic Model for Semistructured Data. In *Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, 1998.

[BFS00] Peter Buneman, Mary Fernandez, and Dan Suciu. UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. *The VLBD Journal*, 9(1): 76-110, 2000.

[BMPW98] Sergey Brin, Rajeev Motwani, Lawrence Page, and Terry Winograd. What Can You Do with a Web in Your Pocket? *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 21(2): 37-47, June 1998.

[Bro98] Gord Brown. IHTML 2: Design and Implementation. In *Proceedings of the 11th International Symposium on Languages for Intensional Programming*, pages 1-13, 1998.

[Bro98a] Gord Brown. Intensional HTML 2: A Practical Approach. Master's Thesis, Department of Computer Science, Univerity of Victoria, Canada, 1998.

[Bun94] Harry Bunt. Context and Dialog Control. *THINK Quarterly*, 3(1): 19-31, 1994.

[Bun97] Peter Buneman. Semistructured Data (Tutorial). In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'97)*, pages 117-121, Tucson, Arizona, 1997.

[CAW98] Sudarshan Chawathe, Serge Abiteboul, and Jennifer Widom. Representing and Querying Changes in Semistructured Data. In *Proceedings of the 14th International Conference on Data Engineering*, Orlando, Florida, February 1998.

[CAW99] Sudarshan Chawathe, Serge Abiteboul, and Jennifer Widom. Managing Historical Semistructured Data. *Theory and Practice of Object Systems*, 24(4): 1-20, 1999.

[CCD+99] Stefano Ceri, Sara Comai, ErnestoDamiani, Piero Fraternali, Stefano Paraboschi, and Letizia Tanca. XML-GL: a Graphical Language for Querying and Restructuring XML Documents. In *Systemi Evaluti per Busi di Dati (SEBD'99)*, pages 151-165, Como, Italy, 1999.

[CD97] S. Chaudhuri, and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1), March 1997.

[CGH+94] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *Proceedings of the Conference on Information Processing Society of Japan (IPSJ'94)*, pages 7-18, Tokyo, Japan, October 1994.

[CH98] William W. Cohen, and Haym Hirsh. Joins that Generalize: Text Classification Using WHIRL. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, 1998.

[Cha99] Sudarshan S. Chawathe. Describing and Manipulating XML Data. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 22(3): 3-9, September 1999.

[CHS+95] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. In *Proceedings of the International Workshop on Research Issues in Data Engineering*, March 1995.

[CK00] Guanling Chen, and David Kotz. A Survey of Context-Aware Mobile Computing Research. Dartmouth College Computer Science Technical Report TR2000-381, November 2000. `ftp://ftp.cs.dartmouth.edu/TR/TR2000-381.ps.Z`

[CL97] Patrick Chan, and Rosanna Lee. *The Java Class Libraries: An Annotated Reference*. Addison-Wesley, 1997.

[Coh98] William W. Cohen. Integration of Heterogeneous Databases Without Common Domains Using Queries Based on textual Similarity. *ACM SIGMOD Conference*, June 1998.

[Coh98a] William W. Cohen. The WHIRL Approach to Integration: An Overview. *AAAI Workshop on AI and Information Integration*, 1998.

[Coh98b] William W. Cohen. A Web-based InformationSystem that Reasons with Structured Collections of Text. In *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, Minneapolis, May 1998.

[Coll96] G. Colliat. OLAP, Relational, and Multidimensional Database Systems. *SIGMOD Record*, 25(3), September 1996.

[CRF00] Don Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. In *Proceedings of the 3rd International ACM SIGMOD Workshop on the Web and Databases (WebDB 2000)*, Dallas, Texas, May 2000.

[CS00] Sophie Cluet, and Jérôme Siméon. YATL: a Functional and Declarative Language for XML. Draft Manuscript, May 2000. `http://www-db.research.bell-labs.com/user/simeon/icfp.ps`

[DBJ99] C. Dyreson, M. Böhlen, and C. Jensen. Capturing and Quering Multiple Aspects of Semistructured Data. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 290-301, 1999.

[DFF+99] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, David Maier, and Dan Suciu. Querying XML Data. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 22(3): 10-18, September 1999.

[DFF+99a] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A Query Language for XML. In *Proceedings of the 8th International World Wide Web Conference (WWW8)*, Toronto, Canada, May 1999.

[DOQT01] E. Damiani, B. Oliboni, E. Quintarelli, and L. Tanca. Modeling Users' Navigation History. In *Proceedings of the Workshop on Intelligent Techniques for Web Personalization*, Seatle, USA, August 2001.

[DS96] Debabrata Dey, and Sumit Sarkar. A Probabilistic Relational Model and Algebra. In *ACM Transactions on Database Systems (TODS'96)*, 21(3): 339-369, September 1996.

[DS98] Debabrata Dey, and Sumit Sarkar. PSQL: A Query Language for Probabilistic Relational Data. In *Data & Knowledge Engineering (DKE'98)*, 28(1): 107-120, 1998.

[Dyr01] Curtis E. Dyreson. Observing Transaction-Time Semantics with TTXPath. In *Proceedings of the 2nd International Conference on Web Information Systems Engineering (WISE'01)*, Kyoto, Japan, December 2001. IEEE Computer Press, pages 193-202, 2001.

[EBNF] ISO (International Organization for Standardization). *ISO / IEC 14977: 1996 (E)*. Extended Backus-Naur Form (EBNF), 1996.

[EPS03] Antonis Efandis, Kostis Pristouris, and Yannis Stavrakas. A Query Evaluation System for Multidimensional Semistructured Data. To appear in *Proceedings of the 1st Balkan Conference on Informatics (BCI 2003)*, Thessaloniki, Greece, November 2003.

[FDFP95] Adam Farquhar, Angela Dappert, Richard Fikes, and Wanda Pratt. Integrating Information Sources Using Context Logic. In *Proceedings of the AAAI Spring Symposium on Information Gathering from Distributed Heterogeneous Environments*, 1995.

[FFK+98] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Catching the Boat with Strudel: Experiences with a Web-Site Management System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*, Seattle, 1998.

[FFLS97] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A Query Language for a Web-Site Management System. *SIGMOD Record*, 26(3): 4-11, September 1997.

[FFLS98] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. Web-Site Management: The Strudel Approach. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 21(2): 14-20, June 1998.

[Fit93] Melvin Fitting. Basic Modal Logic. *Handbook of Logic in Artificial Intelligence and Logic Programming*, Volume I. Oxford University Press, 1993.

[FLM98] Daniela Florescu, Alon Levy, and Alberto Mendelzon. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Record*, 27(3): 59-74, September 1998.

[Fri97] Jeffrey E.F. Friedl. *Mastering Regular Expressions*. O'Reilly, 1997.

[FSW99] Mary Fernandez, Jérôme Siméon, and Philip Wadler. XML Query Languages: Experiences and Exemplars. Draft Manuscript, Communication to the XML Query W3C Working Group, September 1999. `http://www-db.research.bell-labs.com/user/simeon/xquery.html`

[GG01] Chiara Ghidini, and Fausto Giunchiglia. Local Model Semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence* 127, pages 221-259, 2001.

[GHI+95] Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. Integrating and Accessing Heterogeneous Information Sources in TSIMMIS. In *Proceedings of the AAAI Symposium on Information Gathering*, Stanford, California, March 1995.

[Giu93] Fausto Giunchiglia. Contextual Reasoning. *Epistemologia* (Special Issue on I Linguaggi e le Macchine) XVI, pages 345-364, 1993.

[GM00] F. Grandi, and F. Mandreoli. The Valid Web: an XML/XSL Infrastructure for Temporal Management of Web Documents. In *Proceedings of the 1st International Conference on Advances in Information Systems (ADVIS'02)*, pages 294-303, Izmir, Turkey, October 2000.

[GMW99] Roy Goldman, Jason McHugh, and Jennifer Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. In *Proceedings of the 2nd International Workshop on the Web and Databases (WebDB'99)*, Philadelphia, Pennsylvania, June 1999.

[GP98] Luis Gravano, and Yannis Papakonstantinou. Mediating and Metasearching on the Internet. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 21(2): 28-36, June 1998.

[GPQ+97] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey Ullman, Vasilis Vassalos, and Jennifer Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2): 117-132, March-April 1997.

[Gra02] Fabio Grandi. XML Representation and Management of Temporal Information for the Web-Based Cultural Heritage Applications. *Data Science Journal*, 1(1): 68-83, 2002.

[GRGK97] Venkat N. Gudivada, Vijay V. Raghavan, William I. Grosky, and Rajesh Kasanagottu. Information Retrieval on the World Wide Web. *IEEE Internet Computing*, September-October 1997.

[GS03] Manolis Gergatsoulis, and Yannis Stavrakas. Representing Changes in XML Documents Using Dimensions. To appear in *Proceedings of the XML Database Symposium (XSym 2003)*, in Conjunction with VLDB 2003, Berlin, Germany, September 2003.

[GS98] Fabio Grandi, and Maria Rita Scalas. Extending Temporal Database Concepts to the World Wide Web. In *Proceedings of SEBD'98, National Conference on Advanced Database Systems*, pages 53-70, Italy, June 1998.

[GS98a] C. Ghidini, and L. Serafini. Model Theoretic Semantics for Information Integration. In *Proceedings of the 8th International Conference on Artificial Intelligence, Methodology, Systems, and Applications (AIMSA'98)*, Sozopol, Bulgaria, 1998.

[GS98b] Ghiara Ghidini, and Luciano Serafini. Information Integration for Electronic Commerce. In *Proceedings of the Workshop on Agent Mediated Electronic Trading (AMET'98)*, Minneapolis, 1998.

[GSK+01] Manolis Gergatsoulis, Yannis Stavrakas, Dimitris Karteris, Athina Mouzaki, and Dimitris Sterpis. A Web-based System for Handling Multidimensional Information through MXML. In *Proceedings of the 5th East European Conference on Advances in Databases and Information Systems (ADBIS'01)*, Vilnius, Lithuania, September 2001. Lecture Notes in Computer Science (LNCS) 2151, pages 352-365, Springer-Verlag, 2001.

[GSK01] Manolis Gergatsoulis, Yannis Stavrakas, and Dimitris Karteris. Incorporating Dimensions to XML and DTD. In *Proceedings of the 12th International Conference on Database and Expert Systems Applications (DEXA'01)*, Munich, Germany, September 2001. Lecture Notes in Computer Science (LNCS) 2113, pages 646-656, Springer-Verlag, 2001.

[HKWY96] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. An Optimizer for Heterogenous Systems with NonStandard Data and Search Capabilities. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 1996.

[HLLS97] Rainer Himmeröder, Georg Lausen, Bertram Ludäscher, and Christian Schlepphorst. On a Declarative Semantics for Web Queries. In *Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases (DOOD'97)*, Montreux, Switzerland, December 1997.

[HMN+97] L. M. Haas, R. J. Miller, B. Niswonger, M. Tork Roth, P. M. Schwarz, and E. L. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyond Integration. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 1997.

[HTML] The World Wide Web Consortium (W3C). HTML 4.01 Specification, 1999. `http://www.w3.org/TR/html4/`

[HTTP] R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, and T. Berners-Lee. HTTP Version 1.1, 1997. `http://www.ietf.org/rfc/rfc2068.txt`

[ISB95] Tomás Isakowitz, Edward A. Stohr, and P. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design. *Communications of the ACM*, August 1995.

[JAVA] Sun Microsystems, Java, 2003. `http://java.sun.com`

[JCG+92] C. S. Jensen, J. Clifford, S. K. Gadia, A. Segev, and R. T. Snodgrass. A Glossary of Temporal Database Concepts. *SIGMOD Record*, 21(3): 35-43, September 1992.

[JSCR] Netscape Communications Corporation, Javascript, 2003. `http://www.netscape.com/eng/mozilla/3.0/handbook/javascript/`

[Kel97] John Kelly. *The Essence of Logic*. Prentice Hall, 1997.

[KLSS95] Thomas Kirk, Alon Y. Levy, Yehoshua Sagiv, and Divesh Srivastava. The Information Manifold. In *Proceedings of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments*, Stanford, California, March 1995.

[KR88] Brian Kernighan, and Dennis Ritchie. *The C Programming Language*. Prentice Hall, 1988.

[KS95] David Konopnicki, and Oded Shmueli. W3QS: A Query System for the World-Wide Web. In *Proceedings of the 21st International Conference on Very Large DataBases (VLDB'95)*, Zurich, Switzerland, 1995.

[LIN] LINUX Online, 2003. `http://www.linux.org/`

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering Queries Using Views. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, San Jose, California, 1995.

[LSK95] Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data Model and Query Evaluation in Global Information Systems. *Journal of Intelligent Information Systems*, Special Issue on Networked Information Discovery and Retrieval, 1995.

[LSS96] Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. A Declarative Language for Querying and Restructuring the Web. In *Proceedings of the 6th International Workshop on Research Issues in Data Engineering (RIDE'96)*, New Orleans, February 1996.

[MAG+97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. LORE: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3): 54-66, September 1997.

[Mai98] David Maier. Database Desiderata for an XML Query Language, 1998. `http://www.w3.org/TandS/QL/QL98/pp/maier.html`

[Mal98] Susan Malaika. Resistance is Futile: The Web Will Assimilate Your Database. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 21(2): 4-13, June 1998.

[MGSI01] Theodoros Mitakos, Manolis Gergatsoulis, Yannis Stavrakas, and Efstathios V. Ioannidis. Representing Time-Dependent Information in Multidimensional XML. In *Proceedings of the 23rd International Conference on Information Technology Interfaces (ITI'01)*, Pula, Croatia, June 2001.

[MGSI01a] Theodoros Mitakos, Manolis Gergatsoulis, Yannis Stavrakas, and Efstathios V. Ioannidis. Representing Time-Dependent Information in Multidimensional XML. *Journal of Computing and Information Technology*, 9(3): 233-238, 2001.

[MM95] John Mylopoulos, and Renate Motschnig-Pitrik. Partitioning Information Bases with Contexts. In *Proceedings of the 3rd International Conference on Cooperative Information Systems (CoopIS'95)*, pages 44-55, Vienna, Austria, May 1995.

[MMAC99] Giansalvatore Mecca, Paolo Merialdo, Paolo Atzeni, and Valter Crescenzi. The (Short) ARANEUS Guide to Web-Site Development. In *Proceedings of the ACM SIGMOD Workshop on the Web and Databases (WebDB'99)*, Philadelphia, Pennsylvania, June 1999.

[MMM96] Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. In *Proceedings of the 1st International Conference on Parallel and Distributed Information Systems (PDIS'96)*, 1996.

[MMM97] Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. *International Journal of Digital Libraries*, 1(1): 54-67, April 1997.

[NJ02] Andrew Nierman, and H. V. Jagadish. ProTDB: Probabilistic Data in XML. In *Proceedings of the 28th International Conference on Very Large DataBases (VLDB'02)*, Hong Kong, China, 2002.

[Nor02] Kjetil Nørvåg. Algorithms for Temporal Query Operators in XML Databases. In *Proceedings of EDBT Workshops*. Lecture Notes in Computer Science (LNCS) 2490, pages 169-183, Springer-Verlag, 2002.

[NP03] Moira C. Norrie, and Alexios Palinginis. Empowering Databases for Context-Dependent Information Delivery. To appear in *Proceedings of the CAiSE'03 Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS'03)*, Austria, June 2003.

[NP03a] M. C. Norrie, and A. Palinginis. From State to Structure: an XML Web Publishing Framework. To appear in *Proceedings of the 15th International Conference on Advanced Information Systems Engineering (CAiSE'03)*, Austria, June 2003.

[OD97] M. A. Orgun, and W. Du. Multi-dimensional Logic Programming: Theoretical Foundations. *Theoretical Computer Science*, 158(2): 319-345, 1997.

[ODMG] Rick Cattell et al. *The Object Database Standard: ODMG-93, Release 1.2*. Morgan Kaufmann Publishers, 1996.

[OQT01] Barbara Oliboni, Elisa Quintarelli, and Letizia Tanca. Temporal Aspects of Semistructured Data. In *Proceedings of the 8th International Symposium on Temporal Representation and Reasoning (TIME'01)*, pages 119-127, 2001.

[Org96] M. A. Orgun. On Temporal Deductive Databases. *Computational Intelligence*, 12(2): 235-259, 1996.

[OS95] Gultekin Özsoyoğlu, and Richard T. Snodgrass. Temporal and Real-Time Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 7(4): 513-532, August 1995.

[OS99] Aris M. Ouksel, and Amit Sheth. Semantic Interoperability in Global Information Systems: a brief introduction to the research area and the special section. *SIGMOD Record*, 28(1): 5-12, 1999.

[Pap02] Michael P. Papazoglou. The Web-Services Phenomenon: Concepts, Technologies, Current Trends, and Research Directions. Tutorial in the 14th International Conference on Advanced Information Systems Engineering (CAiSE'02), Toronto, Canada, May 2002.

[PERL] Perl programming language, 2003. `http://www.perl.org`

[PGU96] Yannis Papakonstantinou, Hector Garcia-Molina, and Jeffrey Ullman. MedMaker: A Mediation System Based on Declarative Specifications. In *Proceedings of the International Conference on Data Engineering*, pages 132-141, New Orleans, February 1996.

[PGW95] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object Exchange Across Heterogeneous Information Sources. In *Proceedings of the International Conference on Data Engineering*, pages 251-260, Taipei, Taiwan, 1995.

[PW93] J. Plaice, and W. Wadge. A New Approach to Version Control. *IEEE Transactions on Software Engineering*, 19(3): 268-276, 1993.

[RDF1] The World Wide Web Consortium (W3C). Resource Description Framework (RDF) Schema Specification, 1999. `http://www.w3.org/TR/PR-rdf-schema`

[RDF2] The World Wide Web Consortium (W3C). Resource Description Framework (RDF) Model and Syntax Specification, 1999. `http://www.w3.org/ TR/REC-rdf-syntax`

[RS97] Mary Tork Roth, and Peter Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In *Proceedings of the 23rd International Conference on Very Large DataBases (VLDB'97)*, Athens, Greece, August 1997.

[SA85] R. Snodgrass, and Ilsoo Ahn. A Taxonomy of Time in Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 236-246, 1985.

[SD02] S. Shang, and C. Dyreson. Adding Valid Time to XPath. In *Proceedings of Database and Network Information Systems (DNIS'02)*. Lecture Notes in Computer Science (LNCS) 2544, pages 29-42, Springer-Verlag, 2002.

[SG02] Yannis Stavrakas, and Manolis Gergatsoulis. Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE'02)*, Toronto, Canada, May 2002. Lecture Notes in Computer Science (LNCS), Vol. 2348, pages 183-199, Springer-Verlag, 2002.

[SGDZ02] Yannis Stavrakas, Manolis Gergatsoulis, Christos Doulkeridis, and Vassilis Zafeiris. Accommodating Changes in Semistructured Databases Using Multidimensional OEM. In *Proceedings of the 6th East European Conference on Advances in Databases and Information Systems (ADBIS'02)*, Bratislava, Slovakia, September 2002. Lecture Notes in Computer Science (LNCS) 2435, pages 360-373, Springer-Verlag, 2002.

[SGDZ03] Yannis Stavrakas, Manolis Gergatsoulis, Christos Doulkeridis, and Vassilis Zafeiris. Representing and Querying Histories of Semistructured Databases Using Multidimensional OEM. To appear in *Information Systems journal*, 2003.

[SGM00] Yannis Stavrakas, Manolis Gergatsoulis, and Theodoros Mitakos. Representing Context-Dependent Information Using Multidimensional XML. In *Proceedings of the 4th European Conference on Research and Advanced Technology for Digital Libraries (ECDL'00)*, Lisbon, September 2000. Lecture Notes in Computer Science (LNCS) 1923, pages 368-371, Springer-Verlag, 2000.

[SGML] ISO (International Organization for Standardization). *ISO 8879: 1986 (E). Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*. First edition, 1986.

[SGR00] Yannis Stavrakas, Manolis Gergatsoulis, and Panos Rondogiannis. Multidimensional XML. In *Proceedings of the 3rd International Workshop on Distributed Communities on the Web (DCW'00)*, Quebec City, Canada, June 2000. Lecture Notes in Computer Science (LNCS) 1830, pages 100-109, Springer-Verlag, 2000.

[SKV98] Yannis Stavrakas, Nikos Karagiannidis, and Panos Vasiliadis. Direct Database Access through WWW Browsers (in Greek). In *Proceedings of the Panhellenic Conference on New Information Technology*, Athens, Greece, 1998.

[SPES03] Yannis Stavrakas, Kostis Pristouris, Antonis Efandis, and Timos Sellis. Implementing a Query Language for Context-dependent Semistructured Data. Submitted for publication, 2003.

[SQL] International Organization for Standardization (ISO). *Information Technology – Database Language SQL*. Standard No. ISO/IEC 9075:1999.

[SR86] Romualdas Skvarcius, and William B. Robinson. *Discrete Mathematics with Computer Science Applications*. The Benjamin / Cummings Publishing Company, 1986.

[SSU95] Avi Silberschatz, Mike Stonebraker, and Jeff Ullman (editors). Database Research: Achievements and Opportunities Into the 21st Century. *Report of an NSF Workshop on the Future of Database Systems Research*, May 1995.

[Suc97] Dan Suciu. Management of Semistructured Data. *SIGMOD Record*, 26(4): 4-7, December 1997.

[Suc98] Dan Suciu. An Overview of Semistructured Data. *SIGACT News*, 29(4): 28-38, December 1998.

[SW00] P. Swoboda, and W. W. Wadge. Vmake and ISE General Tools for the Intensionalization of Software Systems. *Intensional Programming II*, pages 310-320. World Scientific, 2000.

[SWEB] The World Wide Web Consortium (W3C). Semantic Web, 2001. `http://www.w3.org/2001/sw/`

[SZ96] Avi Silberschatz, and Stan Zdonik. Strategic Directions in Database Systems – Breaking Out of the Box. *ACM Computer Surveys*, 28(4): 764-778, December 1996.

[SZ97] Avi Silberschatz, and Stan Zdonik. Database Systems – Breaking Out of the Box. *SIGMOD Record*, 26(3): 36-50, September 1997.

[TACS98] Manos Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nikos Spyratos. Context in Information Bases. In *Proceedings of the 3rd International Conference on Cooperative Information Systems (CoopIS'98)*, New York City, 1998.

[TG95] V. Tsotras, and B. Gopinath. Efficient Management of Time-Evolving Databases. *IEEE Transactions on Knowledge and Data Engineering*, 7(4): 591-607, August 1995.

[The01] Manos Theodorakis. Contextualization: an Abstraction Mechanism for Information Modeling. PhD Thesis, University of Crete, Greece, 2001.

[Ull88] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems*, Vol. 1. Computer Science Press, 1988.

[Ull97] Jeffrey D. Ullman. Information Integration Using Logical Views. In *Proceedings of the 6th International Conference on Database Theory (ICDT'97)*, Delphi, Greece, January 1997.

[URI] T. Berners-Lee, R. Fielding, U. C. Irvine, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax, 1998. `http://www.ietf.org/rfc/rfc2396.txt`

[URL] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL), 1994. `http://www.ietf.org/rfc/rfc1738.txt`

[W3C] The World Wide Web Consortium (W3C), 2003. `http://www.w3.org/`

[Wal97] Norman Walsb. A Guide to XML. *World Wide Web Journal "XML: Principles, Tools, and Techniques"*, 2(4): 97-107, 1997.

[WBSY98] W. Wadge, G. Brown, M. Schraefel, and T. Yildirim. Intensional HTML. In *Proceedings of the 4th International Workshop on Principles of Digital Document Processing (PODDP'98)*, March 1998. Lecture Notes in Computer Science (LNCS) 1481, pages 128-139, Springer-Verlag, 1998.

[Wie92] Gio Wiederhold. Mediators in the Architecture of Future Information Systems. *The IEEE Computer Magazine*, 25: 38-49, March 1992.

[XLIN] The World Wide Web Consortium (W3C). XML Linking Language (Xlink), 1999. `http://www.w3.org/TR/xlink`

[XML] The World Wide Web Consortium (W3C). eXtensible Markup Language (XML) 1.0 (Second edition), 2000. `http://www.w3.org/TR/2000/REC-xml-20001006`

[XNS] The World Wide Web Consortium (W3C). Namespaces in XML, 1999. `http://www.w3.org/TR/REC-xml-names`

[XPTH] The World Wide Web Consortium (W3C). XML Path Language (XPath), 1999. `http://www.w3.org/TR/xpath`

[XPTR] The World Wide Web Consortium (W3C). XML Pointer Language (XPointer), 1999. `http://www.w3.org/TR/xptr`

[XQL] The World Wide Web Consortium (W3C). XML Query Language (XQL), 1998. `http://www.w3.org/TandS/QL/QL98/pp/xql.html`

[XQR] The World Wide Web Consortium (W3C). XML Query Requirements, 2000. `http://www.w3.org/TR/xmlquery-req`

[XQRY] The World Wide Web Consortium (W3C). XQuery 1.0: An XML Query Language, 2002. `http://www.w3.org/TR/xquery`

[XSC0] The World Wide Web Consortium (W3C). XML Schema Part 0: Primer, 2001. http://www.w3.org/TR/xmlschema-0/

[XSC1] The World Wide Web Consortium (W3C). XML Schema Part 1: Structures, 1999. http://www.w3.org/TR/xmlschema-1/

[XSC2] The World Wide Web Consortium (W3C). XML Schema Part 2: Datatypes, 1999. http://www.w3.org/TR/xmlschema-2/

[XSL] The World Wide Web Consortium (W3C). eXtensible Stylesheet Language (XSL) Version 1.0, 2001. http://www.w3.org/TR/xsl

[XSLT] The World Wide Web Consortium (W3C). XSL Transformations (XSLT) Version 1.0, 1999. http://www.w3.org/TR/xslt

[Yil97] Taner Yildirim. Intensional HTML. Master's Thesis, Department of Computer Science, Univerity of Victoria, Canada, 1997.

[YL96] Budi Yuwono, and Dik Lun Lee. WISE: A World Wide Web Resource Database System. *IEEE Transactions on Knowledge and Data Engineering*, 8(4): 548-554, August 1996.

[ZCF+97] C. Zaniolo, S. Ceri, C. Faloutsos, R. Snodgrass, V. Subrahmanian, and R. Zicari. *Advanced Database Systems*. Morgan Caufmann Publishers, San Francisco, California, 1997.

[ZDSG02] Vassilis Zafeiris, Christos Doulkeridis, Yannis Stavrakas, and Manolis Gergatsoulis. An Infrastructure for Manipulating Multidimensional Semistructured Data. In *Proceedings of the 1st Hellenic Data Management Symposium (HDMS'02)*, Athens, Greece, July 2002.

[ZSC99] Vagelis Zorbas, Yannis Stavrakas, and Kostas Chatzinas. Specification of a Mechanism for Direct Access to Databases through WWW. In *Proceedings of the 7th Hellenic Conference on Informatics*, Ioannina, Greece, 1999.

GLOSSARY / ΓΛΩΣΣΑΡΙ

Angle brackets < >	Άγκιστρα
Atomic node	Ατομικός κόμβος
Bag	Σάκος
Basic change operations	Λειτουργίες βασικών αλλαγών
Bisimulation	Αμφιεξομοίωση
Branching time	Διακλαδιζόμενος χρόνος
Browser	Διαφυλλιστής
Canonical form	Κανονική μορφή
Child element	Στοιχείο παιδί
Clause	Όρος
Clause expanded extension	Ανάπτυξη επεκτεταμένου όρου
Clause expansion	Επέκταση όρου
Clause extension	Ανάπτυξη όρου
Coercion	Εξαναγκασμός
Complex node	Σύνθετος κόμβος
Conceptual	Εννοιολογικός
Concurrency	Ταυτοχρονισμός
Conjunction	Συζευξη
Context	Ερμηνευτικό περιβάλλον
Context aggregate function	Συναθροιστική συνάρτηση περιβάλοντος
Context clause	Όρος περιβάλλοντος
Context coverage	Κάλυψη περιβάλλοντος
Context data path	Μονοπάτι δεδομένων περιβάλλοντος
Context edge	Ακμή περιβάλλοντος
Context element	Στοιχείο περιβάλλοντος
Context equal	Ίσο-περιβάλλον
Context expanded extension	Ανάπτυξη επεκτεταμένου περιβάλλοντος

Context node	Κόμβος περιβάλλοντος
Context path expression	Έκφραση μονοπατιού περιβάλλοντος
Context pattern	Πρότυπο περιβάλλοντος
Context qualifier	Περιοριστής περιβάλλοντος
Context specifier	Προσδιοριστής περιβάλλοντος
Context specifier clause	Όρος προσδιοριστή περιβάλλοντος
Context variable	Μεταβλητή περιβάλλοντος
Context wildcard	Μπαλαντέρ περιβάλλοντος
Context-deterministic	Αιτιοκρατικός ως προς το περιβάλλον
Context-driven query	Επερώτηση οδηγούμενη από περιβάλλον
Core language	Γλώσσα πυρήνα
Cross-world query	Δια-κοσμική επερώτηση
Curly brackets { }	Αγκύλες
Data access method	Μέθοδος προσπέλασης δεδομένων
Data-centric	Δεδομενο-κεντρικός
De-contextualization	Απο-περιβαλλοντοποίηση
Default	Εξ ορισμού
Dimension	Διάσταση
Dimension specifier	Προσδιοριστής διάστασης
Disjunction	Διάζευξη
Domain	Πεδίο ορισμού, γνωστικό πεδίο
Empty clause	Κενός όρος
Empty context	Κενό περιβάλλον
Entity edge	Ακμή οντότητας
Entity part	Τμήμα οντότητας
Expanded form	Επεκτεταμένη μορφή
Explicit context	Ρητό περιβάλλον
Facet	Έκφανση
Facet part	Τμήμα έκφανσης
Fixed-point algorithm	Αλγόριθμος σταθερού σημείου
Formalism	Τυποποίηση
Format	Μορφότυπος
Framework	Πλατφόρμα

General path expression	Γενικευμένη έκφραση μονοπατιού
Global	Ολικός
Graph context projection	Περιβαλλοντική προβολή γράφου
Graph context pruning	Περιβαλλοντική περικοπή γράφου
Graph database	Βάση δεδομένων γράφου
Graph de-contextualization	Αποπεριβαλλοντοποίηση γράφου
Graph model	Μοντέλο γράφου
Graphical interface	Γραφικό περιβάλλον
History of an OEM database	Ιστορικό μιας βάσης δεδομένων OEM
Holding facet	Υφιστάμενη έκφανση
Holds under	Υφίσταται κάτω από
Index	Ευρετήριο
Information entity	Πληροφοριακή οντότητα
Information integration	Ενοποίηση πληροφορίας
Infrastructure	Υποδομή
Inherited context	Κληρονομούμενο περιβάλλον
Inherited coverage	Κληρονομούμενη κάλυψη
Interactive	Διαδραστικός
Interface	Διεπαφή
Join	Σύνδεση
Keyword	Λέξη-κλειδί
Least fixed point	Ελάχιστο σταθερό σημείο
Legal	Επιτρεπόμενος
Link	Σύνδεσμος
Mark-up language	Γλώσσα επισημείωσης
Mediator	Μεσολαβητής
Metadata	Μεταπληροφορία
Middleware tier	Ενδιάμεσο διάζωμα
Module	Μονάδα
Multidimensional attribute	Πολυδιάστατο γνώρισμα
Multidimensional Data Graph	Πολυδιάστατος Γράφος Δεδομένων
Multidimensional DTD / MDTD	Πολυδιάστατο DTD / MDTD
Multidimensional element	Πολυδιάστατο στοιχείο

Multidimensional entity	Πολυδιάστατη οντότητα
Multidimensional node	Πολυδιάστατος κόμβος
Multidimensional OEM / MOEM	Πολυδιάστατο OEM / MOEM
Multidimensional paradigm	Πολυδιάστατο παράδειγμα
Multidimensional Query Language / MQL	Πολυδιάστατη Γλώσσα Επερωτήσεων / MQL
Multidimensional semistructured data	Πολυδιάστατα ημιδομημένα δεδομένα
Multidimensional XML / MXML	Πολυδιάστατη XML / MXML
Multidimensional XSL / MXSL	Πολυδιάστατο XSL / MXSL
Multifaceted entity	Οντότητα πολλαπλών εκφάνσεων
Multi-tier	Πολυ-διαζωματικός
Notation	Σημειογραφία
Object identifier (oid)	Αναγνωριστικό αντικειμένου
Online	Μέσω δικτύου
Partial reduction	Μερική αναγωγή
Path explicit context	Ρητό περιβάλλον μονοπατιού
Path inherited coverage	Κληρονομούμενη κάλυψη μονοπατιού
Possible worlds	Πιθανοί κόσμοι
Predicate	Κατηγόρημα
Probabilistic	Πιθανοτικός
Process	Διεργασία
Reasoning	Συλλογιστική
Reduction to OEM	Αναγωγή σε OEM
Regular expressions	Ομαλές εκφράσεις
Run-time	Χρόνος εκτέλεσης
Scope	Εμβέλεια
Square brackets [ ]	Τετράγωνα άγκιστρα
Start-tag	Κάρτα-αρχής
String	Στοιχειοσειρά
Strongly connected component	Ισχυρά συνδεδεμένο τμήμα
Stylesheet	Φύλλο διαμόρφωσης
Template	Οδηγός
Temporal OEM snapshot	Χρονικό στιγμιότυπο OEM
Timestamp	Χρονοσφραγίδα

Underlying	Υποκείμενος
Universal clause	Καθολικός όρος
Universal context	Καθολικό περιβάλλον
Valid time	Έγκυρος χρόνος
Validity	Εγκυρότητα
Web	Ιστός
Web server	Εξυπηρετητής Ιστού
Web site	Τόπος του Ιστού
Well-formed	Ορθά-σχηματισμένος
With respect to	Σε σχέση με
World Wide Web	Παγκόσμιος Ιστός
Wrapper	Μεταφραστής
XML parsed data	Γραμματικά δεδομένα της XML