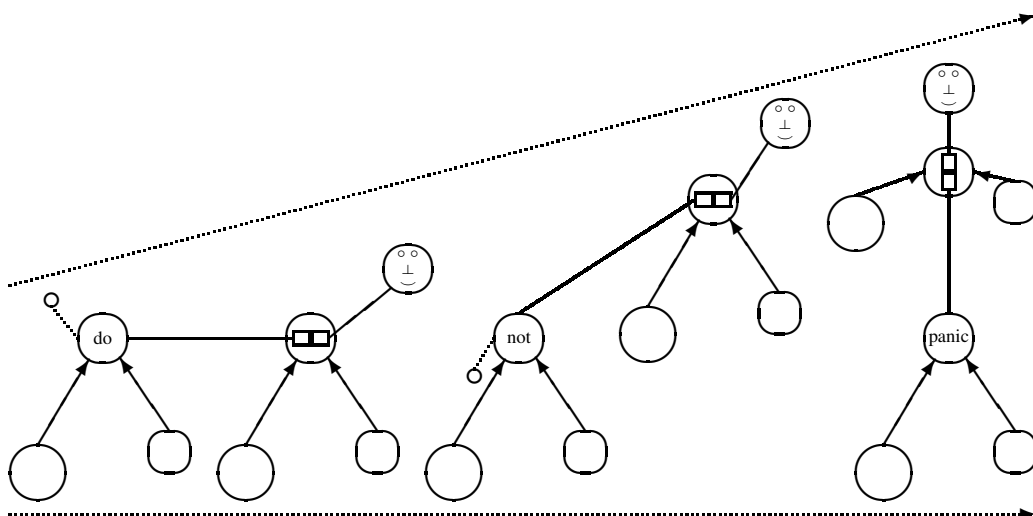


A Theory for Conceptual Modelling of Evolving Application Domains

H.A. (Erik) Proper



A Theory for Conceptual Modelling of Evolving Application Domains

een wetenschappelijke proeve op het gebied
van de
Wiskunde en Informatica

Proefschrift

ter verkrijging van de graad van doctor
aan de Katholieke Universiteit Nijmegen,
volgens besluit van het College van Decanen
in het openbaar te verdedigen op
donderdag 28 april 1994
des namiddags te **3.30 uur** precies

door

Henderik Alex Proper

geboren op 22 mei 1967 te Rheden

Promotor: Prof. dr. E.D. Falkenberg
Co-promotores: Dr. ir. Th.P. van der Weide
Dr. A.H.M. ter Hofstede

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Proper, Henderik Alex

A theory for conceptual modelling of evolving application
domains / Henderik Alex Proper. - [S.l. : s.n.]. - Ill.
Proefschrift Nijmegen. - Met index, lit. opg. - Met
samenvatting in het Nederlands.
ISBN 90-9006849-X
Trefw.: informatiesystemen ; wiskundige modellen.

© 1994 by H.A. Proper, Velp, The Netherlands

All rights reserved. No part of this publication may be reproduced, stored in retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of the author.

A Theory for Conceptual Modelling of Evolving Application Domains

Anything that happens, happens.

*Anything that, in happening, causes something else to
happen, causes something else to happen.*

*Anything that, in happening, causes itself to happen again,
happens again.*

It doesn't necessarily do it in chronological order, though.

From: "Mostly Harmless",
Douglas Adams, Pan Books Ltd.

Manuscriptcommissie:	Prof. dr. G. Engels	Rijksuniversiteit Leiden
	Dr. T.A. Halpin	University of Queensland
	Prof. dr. U.W. Lipeck	Universität Hannover

Dankwoord

Hoewel het schrijven van een proefschrift vooral een solo-arbeid is, is de hulp en steun van anderen onontbeerlijk. Op deze plaats zou ik daarom graag enkele woorden van dank willen uitspreken.

Allereerst zou ik mijn promotor Eckhard Falkenberg willen bedanken. Hij was het die in 1990 het EIS (Evolving Information Systems) project initieerde en mij tot het probleemgebied van evoluerende informatiesystemen introduceerde. Mijn co-promotor Theo van der Weide wil ik bedanken voor zijn diepgaande inhoudelijke betrokkenheid en zijn geduld om mijn ongebreidelde behoefte tot formalisatie in goede banen te leiden. Mijn tweede co-promotor Arthur ter Hofstede zou ik hier willen bedanken voor zijn gewillige oor waaraan ik keer op keer mijn 'hersenspinse-len' kwijt kon. Tevens wil ik hem bedanken voor het zeer minutieus doorlezen van dit proefschrift, zijn commentaar is de interne consistentie van dit proefschrift zeker ten goede gekomen.

Mijn directe collega's binnen het EIS project, Marc McLoughlin en Han Oei wil ik hier bedanken voor de vele interessante discussies aan het begin van het EIS project toen we nog zoekenden in de woestijn waren en ook later toen we allen 'ons weegs' zijn gegaan. Verder wil ik alle collega's van de Afdeling Informatiesystemen bedanken. Zij hebben mij een thuis geboden alwaar mijn ideeën uiteindelijk tot dit proefschrift hebben kunnen evolueren. Hierbij zou ik ook de afstudeerder Coen Burgers willen betrekken.

Een belangrijke bijdrage aan de leesbaarheid van dit proefschrift is geleverd door de proeflezers, van wie ik vooral Denis Verhoef en Miranda Aldham-Breary wil bedanken. Denis heeft kritisch gewaakt over een consistente opzet en uitwerking van de lijn in dit proefschrift. Hiervoor ben ik hem zeer dankbaar. If the English used in this thesis bears a high resemblance to the English language, it is Miranda who I have to thank. Tevens wil ik de leden van de manuscriptcommissie, te weten Gregor Engels, Terry Halpin, en Udo Lipeck, bedanken voor hun commentaar op de concept versie van dit proefschrift.

Gedurende mijn onderzoek ben ik lid geweest van het AXIS (AXiomatische specificatie van InformatieSystemen) promovenditeam, bestaande uit 11 (ex-)promovendi van diverse Nederlandse Universiteiten. Ik wil de andere leden van AXIS hier graag bedanken voor de vele interessante discussies en het delen van de 'promotielast'. Ik zou hier graag de hoop willen uitspreken dat onze samenwerking zich verder zal voortzetten en verdiepen.

Naar mijn mening is het essentieel voor een promovendus om conferenties en workshops te bezoeken, in het bijzonder in het buitenland, teneinde met andere onderzoeksvisies in aanraking te komen. Daar voor reizen geld nodig is, veel geld, wil ik hier graag het NWO, de KUN, en de Koninklijke Shell bedanken voor hun financiële bijdragen.

Laatstelijk zou ik willen dat ik mijn beide ouders zou kunnen bedanken voor hun steun die ze mij gegeven hebben en het geduld dat zij met mij gehad hebben, niet alleen tijdens de laatste 4 jaar, maar gedurende mijn gehele opleiding. Jammer genoeg kan ik nog slechts mijn moeder bedanken voor dit alles, daar mijn vader door zijn veel te vroege overlijden het universitaire gedeelte van mijn opleiding nauwelijks heeft mogen meemaken. Mede daarom wil ik dit proefschrift aan hem opdragen.

Pa, dit is voor jou!

Contents

Dankwoord	v
1 Introduction	1
1.1 Identification of the Problem Area	1
1.2 Terminological Framework	6
1.3 Evolving Information Systems	12
1.4 Problem Statement	18
1.5 Existing Approaches to Evolving Information Systems	19
1.6 Thesis Outline	20
2 Information Structure Universe	21
2.1 Introduction	21
2.2 Generalised Information Structures	22
2.3 Properties of Information Structures	27
2.4 Filtering a Hierarchy	32
2.5 Conclusions	36
3 Evolution of Application Models	37
3.1 Introduction	37
3.2 Modelling Evolution	38
3.3 Generalised Application Models	43
3.4 Application Model Versions	49
3.5 Evolution of Application Models	54
3.6 Conclusions	58
4 A Taxonomy of Update in Information Systems	59
4.1 States of Application Models	59
4.2 The Event Level	62
4.3 The Recording Level	64
4.4 The Correction Level	68
4.5 Overview of the Framework	77

5	Evolving Object Role Models	79
5.1	Applying the General Theory	79
5.2	Informal Introduction to PSM	80
5.3	EVORM Information Structure Universe	85
5.4	EVORM Application Model Universe	92
5.5	Conclusions	99
6	Information Disclosure in an Evolving Environment	101
6.1	Introduction	101
6.2	Semantic Domain	103
6.3	Deriving the Disclosure Schema	112
6.4	Information Descriptors	114
6.5	Language Enrichment	126
6.6	Special Evolution Related Constructs	130
6.7	An Example of Evolution	132
7	Manipulation of Evolving Application Models	135
7.1	Introduction	135
7.2	Informal Introduction to Hydrae	136
7.3	Syntax of Hydrae	138
7.4	Elisa-D Transactions	143
7.5	Epilogue	148
8	Stratified Hypermedia as a Disclosure Mechanism	151
8.1	Introduction	151
8.2	Stratified Hypermedia Architecture	152
8.3	Exploring the Disclosure Schema	154
8.4	The EVORM Hyperindex Layer	155
8.5	The EVORM Hyperbase Layer	164
8.6	Inter Layer Navigation	166
8.7	Conclusions	168
9	Conclusions and Further Research	169
9.1	Problems Statement Revisited	169
9.2	Issues for Further Research	171

A Mathematical Notations	175
A.1 Sets	175
A.2 Functions	175
A.3 Proofs	176
A.4 Tuples	176
B EVORM Graphical Conventions	177
Bibliography	179
Samenvatting	187
Curriculum Vitae	191

Chapter 1

Introduction

There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable.

There is another theory which states that this has already happened.

From: “The Restaurant at the End of the Universe”,
Douglas Adams, Pan Books Ltd.

1.1 Identification of the Problem Area

Nowadays, the financial prosperity of an organisation depends increasingly on its ability to change. Flexibility allows an organisation to be more competitive in the global market, thus improving its chances of survival. This means that organisations must be able to adapt quickly to producing new or different products, changes in the primary process of an organisation that result from ever higher and changing consumer needs. Flexible behaviour of organisations leads to rapidly changing *information need*, any information system supporting such needs must also be highly flexible.

The general problem area of this thesis is outlined in the remainder of this section. We begin with a discussion of the needs of evolving organisations with respect to information systems. Some strategies to cope with evolving information needs are identified. The conclusion is that a new class of information systems is needed: *evolving information systems*. Finally, we provide a structural outline of the chapter using a number of elementary questions, each of which is answered in the remainder of the chapter.

1.1.1 Needs of organisations in an evolving environment

Given the fact that information is gradually becoming a production factor of increasing importance, the need for *flexible information systems* is also increasing. In this respect, it is interesting to note that in the current situation a large percentage of information systems maintenance is carried out because of changes in information needs ([LS80], [Bem87], [BP88]). In case of a highly flexible organisation, this latter percentage is likely to increase even further. The rise of (software!) automation costs is also of increasing concern to many organisations ([VW91]).

To meet the needs of present day *flexible organisations*, information systems that can be adapted to the changing environment in smaller, easier, and more frequent steps than generally is the case, are required. In [Bem87], the adaptability of (information) systems is termed *flexibility in a broader sense*, and the ability of an information system to continue to function in a satisfactory way without having to be changed, although the organisation has, is termed *flexibility in a narrower sense*. Flexible organisations, as discussed above, clearly need information systems which are flexible in a broader sense.

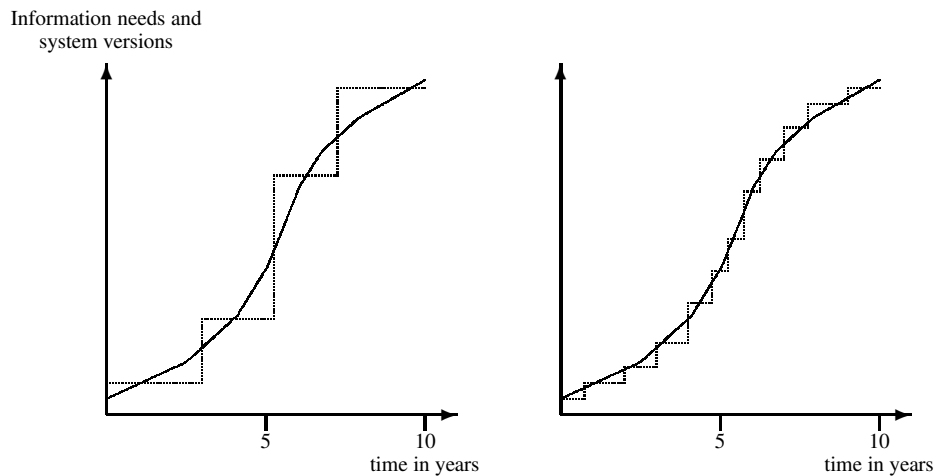


Figure 1.1: Discrepancy between information needs and systems versions

Figure 1.1, presents an intuitive example of what can happen when the information needs of an organisation change with the course of time while the information system is not adapted accordingly. In this figure, which is taken from [Bem87], the dotted lines represent the evolution of an information system, while the solid lines represent the evolution of information needs. The vertical axis denotes the complexity (which can be measured in the number of modelled concepts) of the information need and underlying information system. This figure illustrates that the bigger the time periods between adjustments of the information system are, the bigger the discrepancy between information needs and the information available from the information system. Such a discrepancy can result in an undesirable situation in which users will start to dislike using the information system to fulfill their information needs. Consequently, a rather expensive production factor, the information contained in the information systems, may become idle. The need for information systems which are more flexible, both in the broader and in the narrower sense, than the current generation of information systems has also been argued in e.g. [KM90], [Ari91], [JMSV92]. Two concrete examples of such application domains with rapidly changing information needs are:

Taxes In most nations the (income) tax laws change quite often as they are used both to manage the economy, and to finance government policy.

Software firms developing and maintaining software for the calculation of, say, income taxes for company employees, have to change their software each time the government changes income tax. The change in information needs is caused by the changed laws.

Recent examples of changes in tax laws can be found in many of the nations of the European Community, due to the tax harmonisations brought about by the unification process.

Insurance Most insurance companies change the rules by which policy prices are calculated regularly. These changes are usually intended to improve the competitiveness of the policy pricing, for instance, the no-claims bonus for not claiming damages in the case of car insurances. As a result, the software for calculating the policy prices has to be changed to cope with any extra information required (history of damages claimed), as well as new formulas to perform the calculation. In this case, the changes in the information need are caused by marketing arguments.

In the next subsection, some strategies to meet such evolving information needs are addressed.

1.1.2 Strategies to meet evolving information needs

If organisations change fast, and frequently, the need emerges for an appropriate strategy that allows information systems to evolve to the same extent and at the same pace as the organisations involved. Several approaches to

making information systems more flexible and easier to change can be identified. In this section, we discuss some of these strategies, and relate them to the needs of evolving organisations. This will lead to the definition of the *evolutionary approach* for evolving information systems.

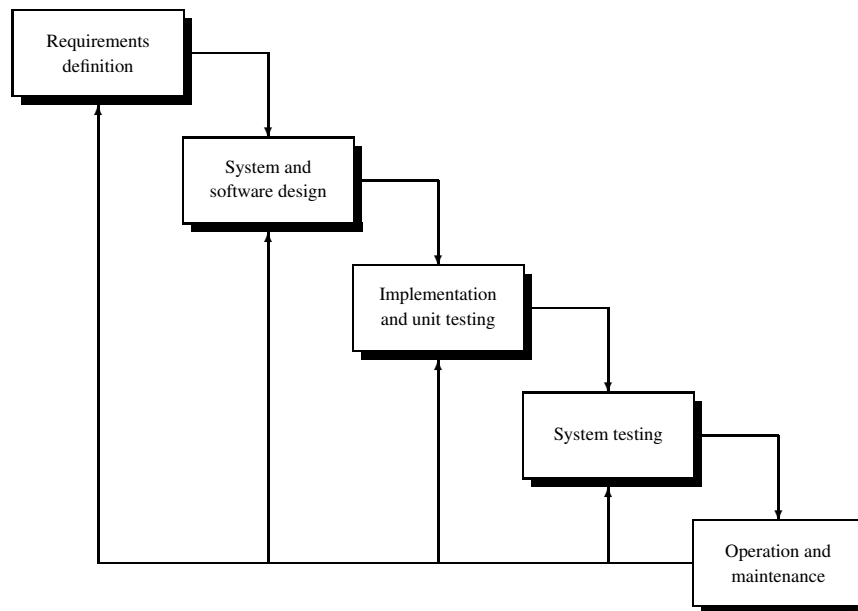


Figure 1.2: Traditional information systems development life cycle

Strategy one, is to try to shorten, or even abolish, the traditional information system development life cycle as illustrated in figure 1.2, which is taken from chapter 1 of [Som89]. A number of ways to do this have been identified in [Lan87], the main focus of these approaches is the concept of *reusability*.

The development of *CASE tools* ([McC89], [OHM⁺88]) and the *software factory* ([Cus89]) are examples of attempts to follow the reuse strategy. Products developed in a CASE tool during the analysis and design phase of an information system (for instance conceptual schemas, data flow diagrams, etc.), can serve as a starting point for the analysis and design phase of the development of a new information system ([McC89], [OHM⁺88]).

The reuse strategy can also be identified in the implementation phase. The possibility of reusing software modules ([Som89]), is clearly intended to shorten the software development life cycle. The reusability of objects classes, and their methods, is also the major claim of the *object-oriented approach* to software development (e.g. [Mey88])

The approaches discussed above, based on reuse, have one important drawback. In the event of a structural change (e.g. the structure of the information stored in the information system) they all require the replacement of the information system, as it is currently in use, by a new system. When performing such a transition from an old to a new system version, processes in the organisation depending on information from the information system may have to be interrupted. When costly data conversions have to be performed to transfer stored information to the new system, the system may not be available for a long period of time.

An alternative approach to the strict incremental development process, often advocated in the traditional life cycle, is a more iterative development process. After the acquisition of an initial set of requirements, an interactive process between designers and users, a first prototype is built on the basis of this set of requirements. This prototype is verified and validated by the future users and the designer(s). The feedback resulting from this verification and validation phase may lead to the development of a new prototype. This process of validation/verification followed by a redesign of the prototype, is repeated until a point is reached where the future users of the information system are satisfied.

This iterative process of information system development, known as *prototyping*, is illustrated in figure 1.3, taken from [Som89]. Prototyping resulted from the observation that in a significant percentage of cases, as argued in e.g.

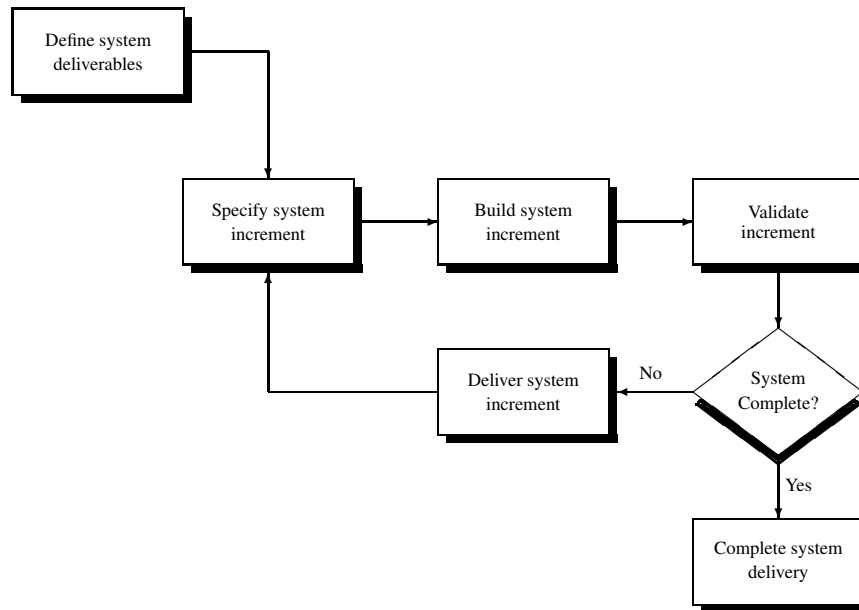


Figure 1.3: Prototyping

[Dav87], requirements for the information systems cannot be established correctly and completely in advance. Even information systems that are based on correctly elicited requirements may be rejected by users, or require substantial and expensive reworking to make them fit the actual needs of the user ([Dav90]), which may even have changed since the start of the (traditional) life cycle of the information system in question. When using the prototyping approach, one still has to replace the information system which is currently being used by the newly developed one (resulting from the prototyping process). A further drawback of the prototyping approach is signalled in [Sol84]. There it is argued that, when using prototyping, converging to a final design for the proposed information system is more time consuming, than is the case in traditional approaches. At present, prototyping is mainly employed as a learning process. After the prototyping phase, the system is re-designed using a traditional development approach.

A new approach for system development, supporting organisations with changing and unclear information needs, is the *evolutionary approach*. The evolutionary approach can be regarded as an advanced kind of prototyping, in which case the prototyping phase is not restricted to the development phase of an information system alone, but is also followed in the *operation* and *maintenance phase*. As a result, the border between these phases in a system's life cycle is blurred. This approach has been discussed (under different names) in [Gil88] as '*evolutionary delivery*', in [BP88] as '*the spiral model*', and in [Bas90] as '*iterative enhancement*'. The signalled drawback of prototyping does not hold for the evolutionary approach, as the developed information system has to be delivered in incremental steps, forcing an early consensus.

The evolutionary approach can be characterised by an iterative life cycle having the length of an organisation's existence. This approach is illustrated in figure 1.4 (taken from [Som89]). In the evolutionary approach, there is no essential difference between the development and maintenance phase of information systems. In both phases, the information system evolves by a sequence of adjustments of the actual information system, which have become necessary due to changes in the organisation and/or changes in the (user) requirements. It should be noted that with the current information system development tools, each of these adjustments require the replacement of the running system by a new system, and may require costly data conversions. Additionally, the system update cannot always be performed automatically.

The DAIDA environment is the first software development environment that actually supports the evolutionary approach ([JMSV92]), however, DAIDA does not support automatic updates of the information system. For an evolutionary approach, in particular when system updates must be performed automatically, a new class of information systems is required. This new class of information systems will be referred to as: *evolving information systems*.

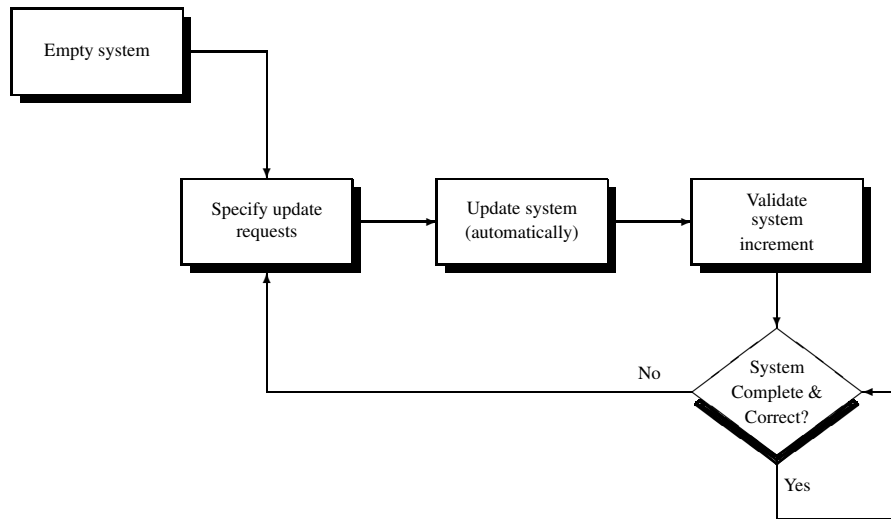


Figure 1.4: Evolutionary approach

1.1.3 Defining Evolving Information Systems

When introducing a new class of information systems, some questions come to mind. In this subsection, we list these questions, and point out where these questions are answered, thus providing an overview of the contents of this chapter.

What is an evolving information system?

The requirements of an *evolving information system*, to provide criteria to decide whether any given information system can be deemed an evolving information system or not, are stated in subsection 1.3.1.

A broad architecture of evolving information systems is discussed in subsection 1.3.2, which also provides a definition of evolving information systems in terms of their components.

Why are evolving information systems needed?

This question has been addressed in the previous subsections.

What research needs to be conducted to establish such systems?

In subsection 1.3.3 objectives for the development of (generalised) evolving information systems are listed, providing a crude research agenda for evolving information systems.

What part of this research is conducted in this thesis?

The answers to the previous three questions taken together provide the relevance, and extent of the research involved in the development of evolving information systems. The research addressed in this thesis is outlined in section 1.4.

What research has already been done?

Related research efforts are discussed in section 1.5. As the field of evolving information systems is rather new, we also look at related research fields, such as historical or temporal information systems, and version management.

To give proper answers to the above questions, a terminological framework needs to be provided, resulting in a better understanding of the *way of thinking* employed in this thesis.

1.2 Terminological Framework

Before stating the requirements for evolving information systems, we introduce two complementary terminological frameworks. The first framework structures the aspects that can be distinguished within an information system development method. This framework is used to position the research needed in the development of evolving information systems, and associated development and maintenance methods.

In the second framework, we consider information systems from an information retrieval perspective, and argue that a traditional information system is essentially a special case of an information retrieval system. This framework allows us to put the requirements and architecture for an evolving information system into a broader perspective. Whereas the first framework covers the modelling process and the involved models, the second framework deals with the final product, the information system itself.

1.2.1 Information systems modelling

In the development of an information system, several methods are used that cover different phases and aspects of the development process. To gain a better understanding of the distinct aspects of these methods a framework is introduced. This framework is used to dissect an information system development method into a series of aspects.

Thus far, we have used the term *information system* without providing a proper definition of the term. In [Ver89], [RV90] two classes of information systems are defined. An *information system in the broader sense* includes all informational aspects of an organisation, whereas an information system in the *narrow sense* only covers computerised aspects. Whenever the term information system is used in this thesis, we use it in the narrow sense.

1.2.1.1 Information systems modelling methods

Figure 1.5 depicts a framework that provides a structured view of modelling methods. It makes the distinction between a way of thinking, a way of working, a way of modelling, a way of communicating, and a way of supporting. The framework is based on the framework originally presented in [WH90] and [SWS89], and elaborated further in [Wij91] and [Ver93a]. In this thesis, we propose an extension of the framework with the notion of *way of communicating* (see also [PW94] and [PW95b]). In the resulting framework, a modelling method is dissected into the following aspects:

1. The *way of thinking* verbalises the assumptions and viewpoints of the method on the kinds of problem domains, solutions and modellers. This notion is also referred to as *die Weltanschauung* ([Sol83]), *underlying perspective* ([Mat81]) or *philosophy* ([Avi95]).
2. The *way of working* structures the way in which an information system is developed. It defines the possible tasks, including sub-tasks, and ordering of tasks, to be performed as part of the development process. It furthermore provides guidelines and suggestions (heuristics) on how these tasks should be performed.
3. The *way of modelling* provides an abstract description of the underlying modelling concepts together with their interrelationships and properties. It structures the models which can be used in the information system development, i.e. it provides an abstract language in which to express the models.
4. The *way of controlling* deals with managerial aspects of information system development. It includes such aspects as human resource management, quality and progress control, and evaluation of plans, i.e. overall project management (see [Ken84] and [Sol88]).
5. The *way of supporting* of a method, refers to the support of the method by (possibly automated) tools. A generally accepted name for computer based tools for methods is: *CASE tools*, see for instance [McC89].
6. The *way of communicating* describes how the abstract notions from the *way of modelling* are visualised (communicated) to human beings, for instance in the style of a conceptual language (such as LISA-D [HPW93]). Usually, the way of communicating uses a graphical notation. It may very well be the case that different methods are based on the same *way of modelling*, and yet use different graphical notations.

The arrows in figure 1.5 should be interpreted as: aspect x supports aspect y .

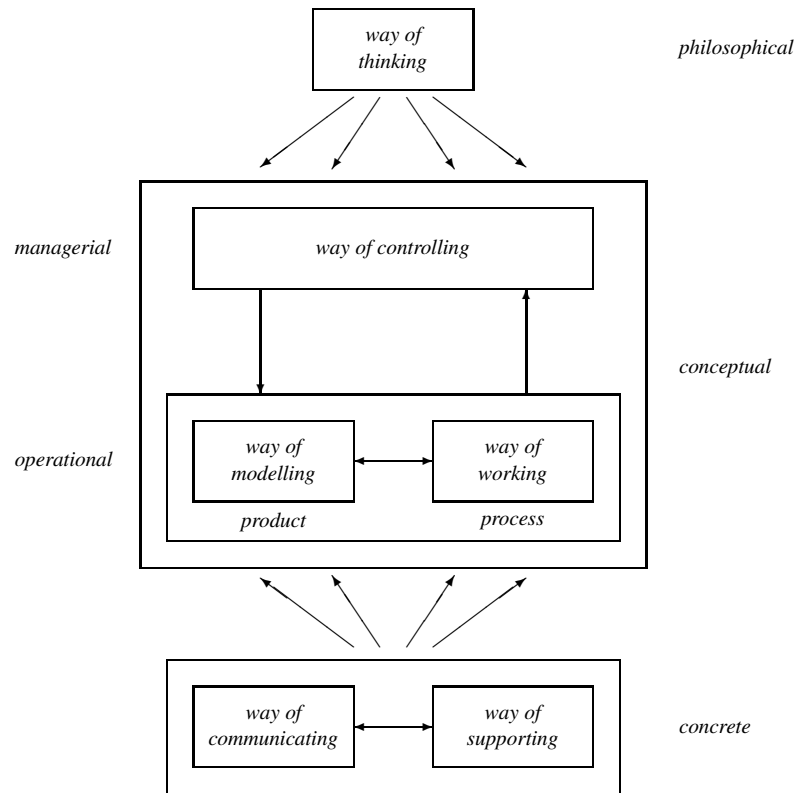


Figure 1.5: The aspects of a method

The combination of a way of modelling and communicating is also referred to as modelling technique. In [Hof93] five requirements for information modelling techniques are formulated. Techniques for information modelling focus on *what* an information system does, or should do, rather than on *how* it should do so. A good information modelling technique, should provide well-defined *formal semantics* for models, i.e. models with an unequivocal meaning, and have sufficient *expressive power* to describe the universe of discourse. The resulting models should be on a *conceptual level*, additionally, since the resulting models play a crucial role in communication with domain experts, the way of communicating should be such that these models are *comprehensible*. Insight into information models may be improved, if they can be executed. Finally, one would like to ‘program’ an (evolving) information system on a conceptual level. Therefore, the models should also be *executable*.

1.2.1.2 A complete specification of the universe of discourse

When developing an information system, a formal description of the universe of discourse under consideration has to be made. The *universe of discourse* is defined as the area of the real or postulated world that is modelled by an information system ([ISO87]).

Remark 1.2.1

In [Hof93], the term application domain is used as an alternative for the term universe of discourse. However, we prefer to consider an application domain to be a more generic term. As a result of this, a universe of discourse corresponds to exactly one application, whereas an application domain refers to a group of applications (and associated universes of discourse) sharing a common property. □

A complete specification of a universe of discourse, referred to as the *application model* ([FOP92a]), typically contains the following components ([ISO87], [OHM⁺88]):

1. An intensional description of the set of states, also referred to as the *information structure*. This set of states modelled by the information structure is defined as the semantics of the information structure.
2. A further restriction of the set of states by *static constraints*.
3. An intensional description of the set of transitions that can be performed by the system, usually as a set of *action specifications*.
4. A restriction of the set of possible transitions (between valid states) by *dynamic constraints*. Quite often, this component is incorporated in the *action specifications*, in the form of pre- and post-conditions.
5. An extensional specification of the current state of the universe of discourse, i.e. the *population*, or *information base*.

The notion of states of a universe of discourse, used in the above descriptions, has been used before in [FN85], [De 88] and [WW88]. The components listed above, are related to each other in the following way. A population conforms to the underlying information structure. Usually not all possible populations of the information structure correspond to proper states of the universe of discourse, and therefore have to be excluded. For this purpose, static constraints are employed, which simply exclude invalid populations. Analogously, dynamic constraints are used to exclude transitions of states that do not correspond to transitions in the universe of discourse. By forbidding transitions, dynamic constraints may render states unreachable from the initial state. These unreachable states are also not valid. The refinement of proper states (populations) of the information structure of a given universe of discourse is illustrated in figure 1.6, taken from [HW93].

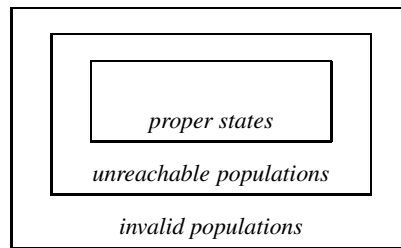


Figure 1.6: The classification of populations

Finally, the set of action specifications describes the changes that can be performed on the population of the information structure, as a response to update requests of the user.

Remark 1.2.2

The style of describing, used above, is known as the superset technique. This technique is typically used in cases where a precise description is cumbersome. In these cases, one looks for an easy way to describe a superset (the information structure), and excludes invalid elements by imposing constraints on the elements of the superset. For more information about formalisation in this context, see [HW92]. □

Using the components of a specification of a universe of discourse discussed above, an application model can be subdivided into further sub-models, resulting in the hierarchy of models shown in figure 1.7. The components are identified by:

1. The *world model* encompassing the combination of information structure, both static and dynamic constraints, and a population conforming to these requirements.
2. The *action model* is a set of action specifications that describe the transitions that can be performed by the system.

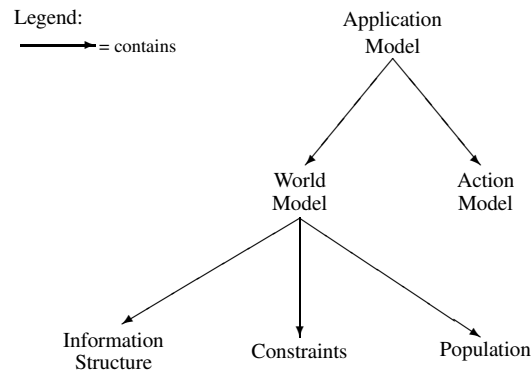


Figure 1.7: A Hierarchy of Models

Following [ISO87], we refer to the combination of information structure and constraints as the *conceptual schema*.

Traditionally, a world model is given as a data model and a population conforming to this data model. Usually, the data model is given as a model conforming to a data modelling technique such as:

NIAM ([NH89], [Win90]), FORM ([Hal89], [HO92]), INFOMOD ([JG87]),
 ER ([Che76])

or for more complex applications, a modelling techniques such as:

IFO ([AH87]), EER ([HNSE87], [HE92]), ER Set Model ([EWH85]),
 PSM ([HW93], [HPW92b], [HPW92c]).

An action model is usually modelled by petri-net like specifications such as:

ExSpect ([HSV89], [HV91]), Data Flow Diagrams ([You89], [TP89]),
 Task Structures ([WHO92], [HN93]),

or a database language like SQL.

Ideally, the application model is modelled as one single model, using a coherent set of modelling techniques. Such a single coherent model will contain all identified parts of the application model as sub-models. Different frameworks exist for the different perspectives from which universes of discourse may be modelled. Recent examples of such frameworks for *data* intensive application domains can be found in: [Stu92], [Hof93], and for *processing* intensive domains in [Ver93b], [Ram94].

The application model of a universe of discourse is denoted in terms of object types, constraints, instantiations, action specifications, etc. The term *application model element* is employed ([FOP92a]) as a collective noun for these modelling concepts.

1.2.1.3 The corpus evolutionis

The complete specification of a universe of discourse should be allowed to change with the course of time. In most traditional information systems, the application model can only be partially changed. The part of an information system allowed to change with the course of time, will be referred to as the '*corpus evolutionis*'. Just as '*corpus delicti*', meaning the *substance of crime*, '*corpus evolutionis*' is the *substance of evolution*.

In most traditional information systems the corpus evolutionis is restricted to the population. In general, the corpus evolutionis only encompasses the set of facts obeying a fixed (conceptual) schema with a fixed set of constraints. In other words, update of the conceptual schema, and consequently the internal data base schema, constraints, and specifications of dynamic aspects, i.e. the action model, are not as yet supported by these traditional information systems. Nevertheless, some traditional information systems do, to a limited extent, support modifications of other components from the application model. For example, adding a new table in an SQL system is easily done, however,

changing the arity of a table, or some of its attributes, will result in a time consuming table conversion, which may also lead to the loss of the information contained in the old table!

1.2.2 Information retrieval paradigm

This subsection discusses our way of thinking with regard to information (retrieval) systems and their context. In accordance with modern approaches to *information retrieval systems*, information systems can be looked upon as information retrieval systems ([Bru93]). As traditional information retrieval systems lack the ability to handle *structured-information* adequately, the *information retrieval paradigm* presented in [Bru93] and [BW92b] is extended to support structured information. We do this by extending the paradigm with the notion of *universe of discourse* and *conceptual description*. Later, this *structured-information retrieval paradigm* will be extended even further with the notion of *update*, resulting in the *evolving information system paradigm*.

1.2.2.1 Traditional information retrieval systems

We will first focus on the historical background of the information, i.e. the *information disclosure* problem. The information disclosure problem starts with an individual or company with an *information need* they wish to fulfill. The information need is typically made more concrete by an *information request* designated q , in some (formal) language, and should be as good as possible a description of the information need. The information request is then passed on to an automated system, or human intermediary, who will try to fulfill the information request using the information stored in the system. This is illustrated in the *information disclosure*, or *information retrieval paradigm*, presented in figure 1.8, and taken from [BW92b].

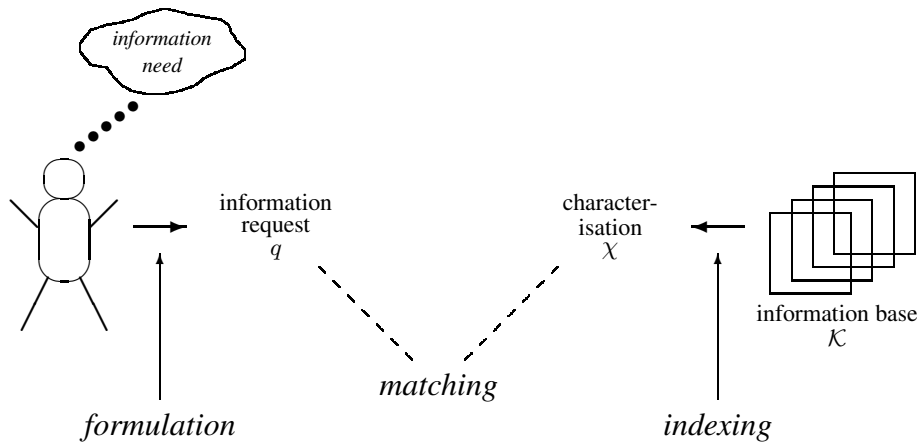


Figure 1.8: The information retrieval paradigm

Retrievable information is modelled as a set \mathcal{K} of *information objects* constituting the *information base* (or population); in a *document retrieval system* the information base will be a set of documents ([SM83]), while in case of a traditional information system the information base will contain a set of facts conforming to a *conceptual schema* ([ISO87]). Each information object o is *characterised* by a set of descriptors $\chi(o)$ that facilitates its disclosure. These descriptors are typically drawn from a *descriptor language*. The characterisation of the information objects is carried out by a process referred to as indexing.

In a traditional information system all stored objects (the population, or information base) can be identified by a set of (denotable) values, the identification of the object, for example an address consists of a city name, street name and house number. Thus, the characterisation of information objects in such systems is provided as an *identification mechanism* based on the underlying conceptual schema ([Hof93]).

Information disclosure is typically driven by a process referred to as *matching*. In document retrieval applications this *matching process* tends to be rather complex, as the characterisation of documents is known to be a hard problem ([Mar77], [Cra86], [Sal89]). In traditional information systems this process is less complex, as the facts in the information base have a more clear characterisation (the identification). In this case, the identification of the information objects (facts) is simply related to the formulation q of the information need of the user, by some (formal) query language.

1.2.2.2 Structured-information retrieval

The information retrieval paradigm presented in figure 1.8 originates from the late fifties. Most present day applications, however, do not deal with amorphous objects. Instead, they have to deal with information objects that have a structure. In the field of document retrieval, standards like SGML ([ISO86]) and ODA ([ISO89]) provide more structured ways to look at documents.

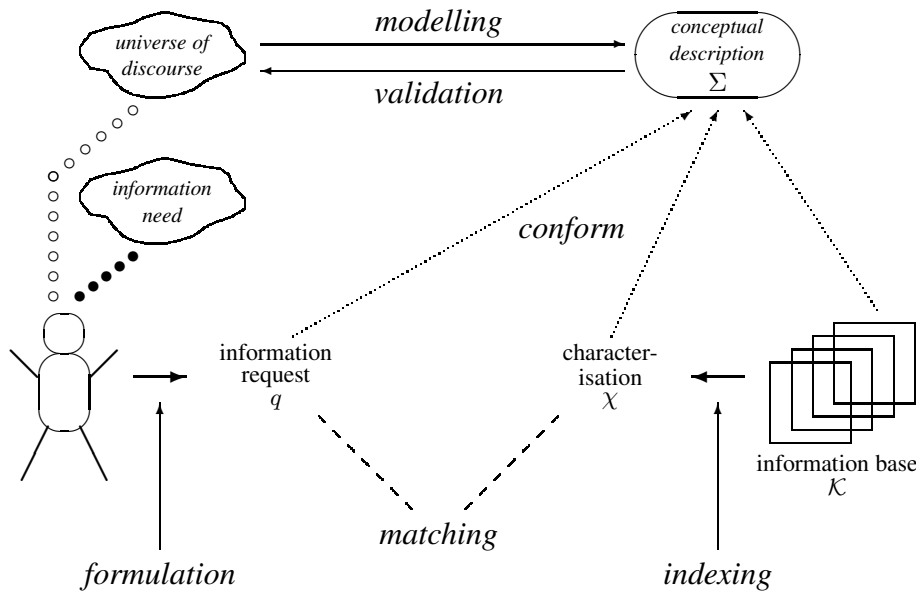


Figure 1.9: The structured-information retrieval paradigm

This more structured way of looking at information objects is supported by a grammatical structure (usually a context-free grammar) for the information objects, the *conceptual description*. In traditional information systems, this *conceptual description* has always been inherently available through the underlying conceptual schema. The information retrieval paradigm, presented in figure 1.8 can now be extended to the *structured-information retrieval paradigm*, supporting information with an explicit structure. This new paradigm is depicted in figure 1.9. The information request (q), the characterisation ($\chi(o)$) of information objects, as well as the information objects ($o \in \mathcal{K}$) must conform to a (formal) language, based on the conceptual description (Σ). This conceptual description is derived from the universe of discourse in a process referred to as modelling, and its correctness must be validated against the universe of discourse.

Remark 1.2.3

It is interesting to note that figure 1.9 captures information retrieval systems and traditional information systems, as well as expert systems ([HRWL83], [BS84], [LG91]) including decision support systems and executive information systems. In expert systems two important components can be distinguished, the knowledge base and the inference engine. The knowledge base captures domain-specific knowledge, whereas the inference engine consists of algorithms for the manipulation of the knowledge present in the knowledge base.

In terms of figure 1.9, the knowledge base is split into an intensional knowledge base, and an extensional knowledge base. The extensional knowledge base corresponds to the information base (and characterisation), containing the known facts and their probabilities. The intensional knowledge base corresponds to the conceptual description. It contains a description of the structure of the stored information, together with a set of production rules for the derivation of new knowledge.

The activities of the inference engine, in response to an information request from the user, correspond directly to the matching process. □

1.2.2.3 A general information retrieval system architecture

The conceptual architecture for information retrieval systems, is based on [Bub86] and [ISO87]. This architecture provides a further refinement of the structured-information retrieval paradigm, as well as of our way of thinking with regards to information (retrieval) systems. The architecture refines the structured-information retrieval paradigm in terms of the components of an information (retrieval) system, and is depicted in figure 1.10.

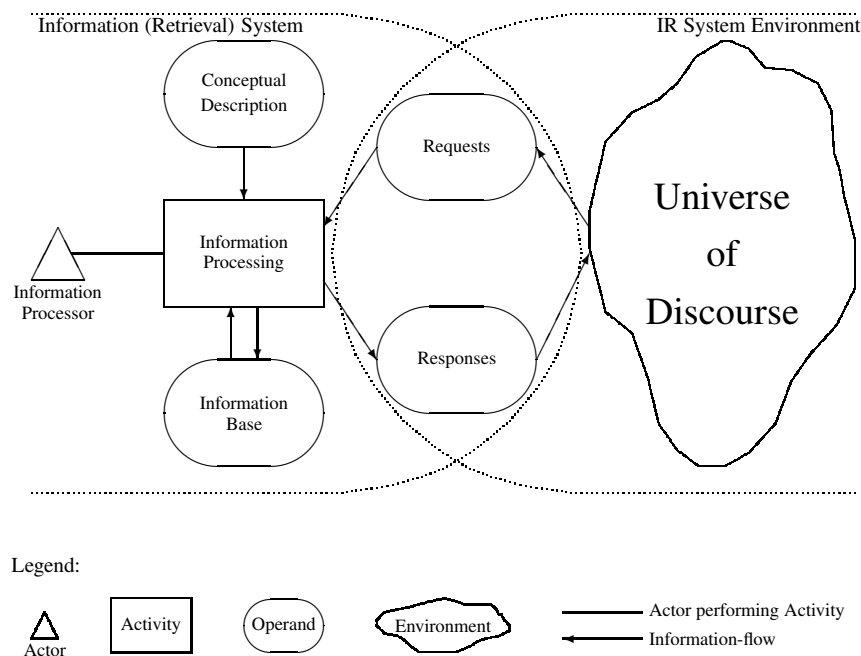


Figure 1.10: Information (Retrieval) System Architecture

The *information processor* performs the information processing activities. In an information system in the narrow sense, the information processor is restricted to a computer, whereas the information processor in an information system in the broader sense may be (partially!) human. Processing involves the acceptance of input messages, either *information requests* or *update requests*. Information requests are matched to the characterisations of the stored objects (for instance by using a disclosure machine as discussed in [Bru93]). Update requests result in appropriate changes of the *information base*, and may cause the initiation of other information processing activities. The information processor may generate output messages, as response to the input messages, which will be received in the environment of the information system. During information processing activities, the information processor uses the conceptual description to maintain the consistency of the information base. Note that, as a result of our arguments, information systems can be regarded as information retrieval systems, and that *conceptual description* is now synonym to *conceptual schema*.

1.3 Evolving Information Systems

In this section we provide a definition of evolving information systems in terms of a set of requirements. A general architecture for evolving information systems is provided.

1.3.1 Requirements of evolving information systems

The requirements of an evolving information system are beyond those of traditional information systems, including *temporal information systems* ([OPF94]). The main requirement of an evolving information system is that it is able to evolve to the same extent and at the same pace as the underlying universe of discourse, without the need to interrupt the processes depending on the information system, i.e. on-line. Given this general requirement, some more detailed requirements are formulated below:

1. The system should provide an adequate disclosure mechanism for the retrieval of *all* stored information.

This requirement may seem trivial and one not limited to evolving information systems. Nevertheless, a good disclosure mechanism becomes essential when the underlying (information) structure of the stored information changes with the course of time, rendering traditional query languages such as SQL inadequate, since they require the user to have a good understanding of the (internal) table structure of the stored information. It should be obvious that techniques used in information retrieval systems can be applied for this purpose.

2. The information system should allow *update* of all information depending on the specific universe of discourse ([ISO87]) and maintain a set of well-formedness rules.

Maintaining these well-formedness rules excludes unwanted steps in the evolution of the corpus evolutionis. The corpus evolutionis in an evolving information system consists of the complete application model.

Although it may seem paradoxical, the universe of discourse can evolve while the application domain itself does not evolve. For instance, when knowledge about the application domain increases the application domain itself does not change, but the universe of discourse does. A concrete example of such an application domain is physics. The domain does not change, but our insight does, hence the universe of discourse evolves.

3. The information system should allow for the correction of all information (previously) recorded in the system.

Information which is recorded in the information system may appear to be (empirically) invalid, an evolving information system makes correction of this invalid information possible. Since most of the information recorded in an information system is entered by humans, errors will occur quite frequently.

4. The system must not *forget* any information recorded in the information system unless explicitly told to. The forget operator corresponds to a delete *sensu proprio*, i.e. all traces should be removed from memory.

The complete history of the application model including that of corrections must be kept, unless a user request or law demands that information should be forgotten (e.g. because of privacy reasons and lack of storage capacity).

5. Any update of the information system should not interrupt the activities of the organisation involved.

An evolving information system should minimise the discrepancy between the information needs of an organisation and the information supplied by the information system, giving rise to a large number of (small) structural changes in the system. For this reason, the information system will be required to remain available to the users of the system during structural changes.

One important consequence of these requirements is that the notion of time must be introduced into the theory of evolving information systems. Two notions of time are distinguished in the literature: *valid time* representing the time at which an event (change of state) occurs in the universe of discourse, and *transaction time* representing the time at which the event is recorded in the information system (see [Bub80], [SA85], [SA86], [Sno90], [RP92]). As an evolving information system must not forget anything unless explicitly directed, recordings and occurrences of events, and corrections to these recordings, must also be recorded. A direct result of this observation is that an evolving information system must support both notions of time ([FOP92a], [FOP92c]). In [SA86], a classification of information systems is made with respect to their support of valid time and transaction time. This classification distinguishes:

1. *Snapshot information systems* not supporting any explicit notion of time.
2. *Historical information systems* supporting valid time.
3. *Roll-back information systems* supporting transaction time, thus allowing for a roll-back in the sequence of recorded transactions.
4. *Temporal information systems* supporting both notions of time.

Following this classification, an evolving information system will be a temporal information system, as both valid and transaction time are supported. It should, however, be noted that not all temporal information systems are evolving information systems. As we have seen in this section, evolving information systems have to meet additional requirements with respect to their corpus evolutionis. As such, evolving information systems form a new class.

Remark 1.3.1

When evolving information systems become available, traditional systems may turn out to be ‘degenerations’ of these evolving information systems, see [FOP92a] and [FOP92c]. Most traditional information systems do not support the notion of time (both recording and event time), they are ‘snapshot’ systems. They reflect information valid at a certain point in time, but lack the ability to preserve the history of the information. □

1.3.2 Architecture for evolving information systems

The structured-information retrieval paradigm does not yet take the evolution of (parts of) the application model into consideration. This paradigm must therefore be extended to the evolving information system paradigm presented in figure 1.11. With this extension, the paradigm also takes the notion of update into consideration. A user may specify an update request, resulting in an update of the application model. The update of the application model may also lead to an update of the characterisation of the components of the application model. More than one application model can be maintained, the current application model and the complete history can be kept.

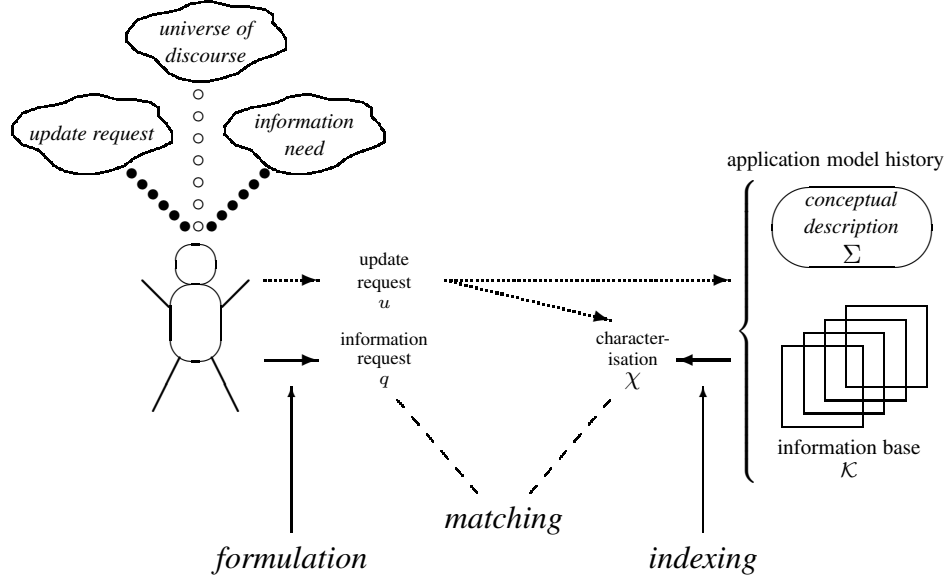


Figure 1.11: Evolving Information Systems Paradigm

The main difference between a traditional information system and an evolving information system, can again be illustrated using the general architecture for evolving information systems depicted in figure 1.12 ([FOP92c]). In an

evolving information system, the only fixed part is the *meta model* (for a more elaborate discussion on meta models, refer to e.g. [Bri90], [BF91]). A meta model is *time-* and *application independent*, it contains (and is restricted to) all rules about the languages used to model the application model and formulate user *requests*. Since the history of the application model is maintained, the information processing activities operate upon the history of the application model as a whole, and not just on one version.

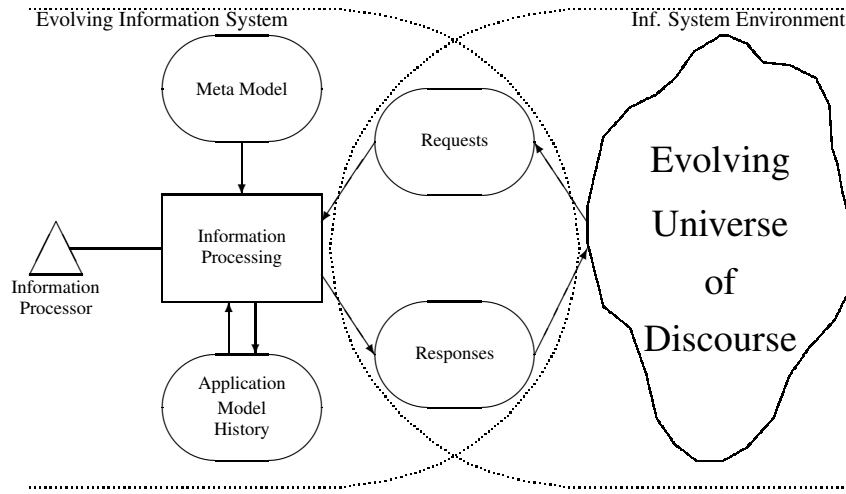


Figure 1.12: Evolving Information System Architecture

We finish this section with an elaborated example of an evolving universe of discourse, which will be used as a running example in this thesis.

Example 1.3.1

As an illustration of an evolving universe of discourse, consider a Record rental store founded in the sixties. A catalogue of available Records is maintained by the store. To keep track of the wear and tear of Records, the number of times a Record is rented is recorded. The information structure and constraints of this universe of discourse are modelled in figure 1.13 in the style of (E)ER. Note the special notation of attributes (Title) using a mark symbol (#) followed by the attribute (# Title).

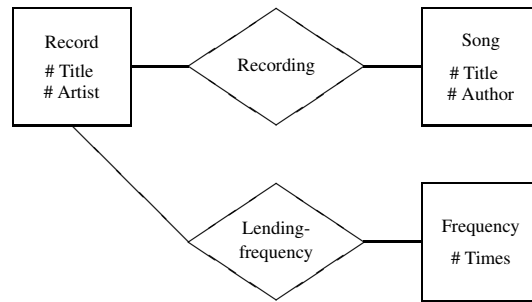


Figure 1.13: The information structure of an Record rental store

An action specification in this example is the rule *Init-freq*, stating that whenever a new Record is added to the selection available in the store, its lending frequency must be set to 0:

```

ACTION Init-freq =
  WHEN ADD Record:  $x$  DO
    ADD Record:  $x$  has Lending-frequency with Frequency: 0
  
```

This action specification is in the style of LISA-D ([HPW93], [HPW94a]). Note that the expression `Frequency: 0` denotes the instance of entity type `Frequency`, which is identified by the number of `Times: 0`.

After the introduction of the compact disc, and its conquest of a sizable share of the market, the rental store became a 'Record and CD rental store'. This leads to the introduction of an object type `Medium` as a generic term for `Record` and `CD`. In the new situation, the registration of songs on `Records` is extended to cover `CDs`. The frequency of lending, however, is not kept for `CDs`, as `CDs` suffer little wear and tear. As a consequence, the application model has evolved to figure 1.14. This requires an update of the typing relation of instances of object type `Record`, which are now instances of both `Records` and `Medium`. Note that this modification can be done automatically.

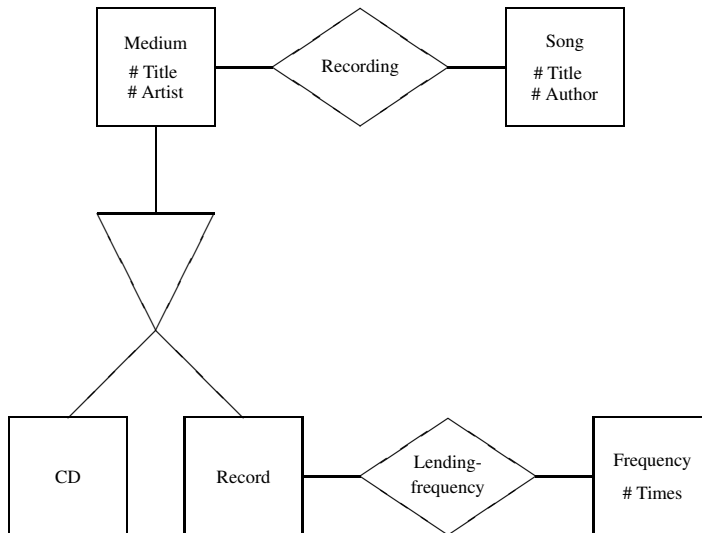


Figure 1.14: The information structure of an Record and CD rental store

The action specification `Init-freq` evolves accordingly, now stating that whenever a medium is added to the selection available at the rental store, its lending frequency is set to 0 provided the medium is a `Record`:

```

ACTION Init-freq =
  WHEN ADD Medium: x DO
    IF Record: x THEN ADD Record: x has Lending-frequency with Frequency: 0
  
```

After some years `CDs` have become more popular than `Records`, consequently, the rental store decides to stop renting `Records` and to become a `CD rental store`. This change in the rental store, leads to the information structure as depicted in figure 1.15. Now the recording quality of songs on `CDs` is relevant for clients, since this quality may differ from song to song on a single `CD`, and furthermore, may for some songs be different for recordings of the song on different `CDs`. Recording quality is added as a (mandatory) attribute to the `Recording` relation. This change in the rental store, leads to the information structure depicted in figure 1.15.



Figure 1.15: The information structure of a CD rental store

As a result of this evolution step, the action specification `Init-freq` can be terminated, since the lending frequency for `CDs` is no longer recorded. The addition of the mandatory attribute `Quality` enforces an update of

the existing population. In this case, in contrast to the previous evolution step, information is added to the old population. This could, for example, be effectuated by the following action:

```
FOR ALL  $r$  IN Recording DO
  ADD Recording  $r$  has Quality 'AAD'
```

□

1.3.3 Generalised evolving information system

Given a meta model for evolving information systems, a management system for these evolving information systems can be developed which is time-invariant and independent of any universe of discourse. Such an environment is a generalised evolving information system, and is referred to as *evolving information management system (EIMS)*. When an evolving information system has to be developed for a particular universe of discourse, an application model describing this domain is built-up and maintained in conformance with the language (ER, NIAM, Task Structures, etc.) defined by the meta model of an EIMS. In this section we discuss the research needed to develop such a management system.

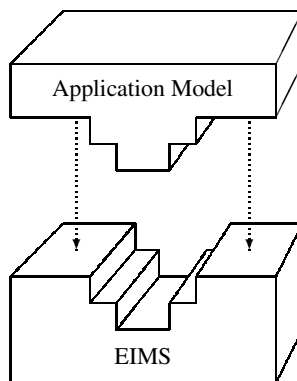


Figure 1.16: The EIMS: independent of any application model

An EIMS should be independent of any universe of discourse. As a consequence, application models describing different domains can be ‘plugged’ into the EIMS. This principle is illustrated in figure 1.16. Furthermore, a management system for evolving information systems should be designed in such a way that it is independent of any software environment, i.e. independent of any database management system and/or operating system. This independency is illustrated in figure 1.17.

During the development of an EIMS, three subobjectives can be distinguished:

1. A meta model and associated language for the specification and maintenance of the so-called *application model* must be designed. This language must be able to support all aspects of evolution, and provide an adequate *disclosure mechanism* for all stored information.
2. An EIMS can then be implemented based on that meta model and language.
3. A suitable procedure for design and maintenance of the application model must be developed.

The use of an EIMS, together with the design and maintenance procedure, in an organisation can be seen as the use of a method for modelling the evolution of organisations. The validity of this statement follows from the following observations, relating the above objectives to the components of a method:

1. The way of thinking of an *evolving information system*, and its associated design and maintenance procedure, is discussed in the previous section.

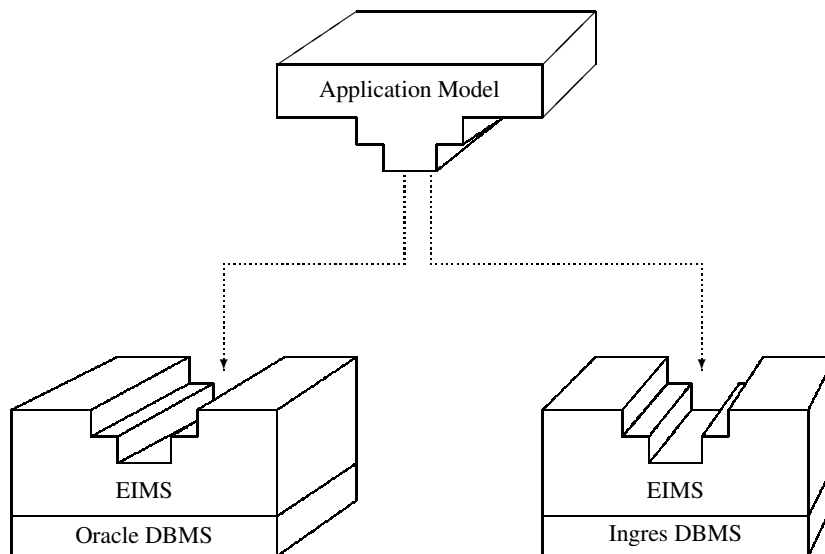


Figure 1.17: The EIMS: independent of any software environment

2. The *way of working* is partially given by objective 3. The exact definition of the way of working depends highly on the way of working of the chosen modelling techniques for the components of the application model.
3. The *way of modelling* is given by objective 1, and depends partially on the modelling techniques chosen for the application model.
4. The *way of controlling* is given by objective 3.
5. The *way of supporting* of an evolving information system is given by the management system, objective 2.
6. The *way of communicating* is given by objective 1, in the form of the modelling techniques used for the application model.

1.4 Problem Statement

Having provided a broad perspective on evolving information systems and their context, consisting of a definition of evolving information systems in terms of their requirements, a discussion on the sub-objectives in the development of an EIMS and their relation to the notion of a method, we are now in a position to state the subject of this thesis.

In this thesis we focus on objective 1, resulting in a way of modelling and way of communicating of a method for the development and maintenance of an evolving information system.

It is our opinion that the way of modelling is the nucleus of any method, and therefore that the development of a method for evolving information systems, should start from this point. The resulting way of modelling and communicating should together form an information modelling technique that conforms to the five requirements stated in subsection 1.2.1: *formal semantics, expressive power, conceptual level, comprehensible, and executable*.

When choosing a way of modelling and communicating for evolving information systems, one should realise that in the last decades a plethora of modelling techniques has been developed for the various components of an application model (see e.g. [Bub86]). In general, these modelling techniques provide only a crude and incomplete description of their syntax, and semantics ([HW92]). This situation has led to *The Methodology Jungle* ([Avi95]). In this situation it is extremely difficult, using an objective criterion such as expressiveness, or a subjective criterion such as suitability, to choose a modelling technique for the application model in an EIMS. Therefore, the first focus in this thesis is:

Development of a general theory for the way of modelling and communicating of evolution of application models. The resulting theory abstracts from intrinsic details of concrete modelling techniques.

To obtain a concrete way of modelling and communicating for evolving information systems, and as a first test-case for the general theory, we will apply the general theory to the integrated set of modelling techniques PSM, LISA-D, Hydra. The data modelling technique PSM ([HW93]) is an extension of PM ([BHW91]), which is in its turn a formalisation of NIAM ([NH89]). PSM has been defined as a generalisation of object role modelling techniques such as:

PM, NIAM, FORM ([HO92]), INFOMOD ([JG87]), ER ([Che76]), EER ([HNSE87]),
ER Set Model ([EWH85]), and IFO ([AH87]).

LISA-D (*Language for Information Structure and Access Descriptions*), is the language (way of communicating) for the formulation of queries, updates (of the population) and PSM information structures. The Hydra ([Hof93]) action modelling technique is an integration of Task Structures ([WHO92], [HN93]). Hydra's formal semantics have been defined in terms of Process Algebra ([BW90a]), and PSM using LISA-D statements as elementary transactions. Hydra can be seen as a generalisation of process/data-flow modelling techniques such as DFD ([You89], [TP89], [BW89]) and ISAC-A schemata ([LGN81]). Applying the general evolution theory for application models to the PSM–LISA-D–Hydra framework thus provides a good test-case, covering a wide range of modelling techniques.

1.5 Existing Approaches to Evolving Information Systems

As stated in subsection 1.3.1, the classification for the incorporation of time in information systems provided in surveys of this area (see e.g. [SA85], [McK86], [Sno90]) makes a distinction between roll-back, historical and temporal information systems; however, all these classes do not as yet take schema evolution into account. For this reason, we proposed a new class: evolving information systems.

Although the first three classes of information systems do not support evolution, we will provide a short overview of the research in this area as some aspects are useful for evolving information systems. In the TEMPORA project ([TLW91], [MSW92]), the ER model is extended with the notion of time by associating a time-stamp-type to those object types for which time-stamping may be relevant, resulting in the ERT model. In TODM ([Ari86]) and ERAE ([DHL⁺85], [DDP93]), similar strategies are followed, extending the relational model with the notion of time. In [CW83], [Tan86], [NA87], [Sno87], [Gad88], [TC90], the relational algebra and SQL are extended with temporal features. These developments make it possible to handle historical data over a (non-varying) underlying information structure. In [LS87], [Saa91] and [Saa88] the focus is on the monitoring of dynamic constraints, i.e. constraints over such historical data. Dynamic constraints restrict temporal evolutions, i.e., state sequences of databases. Historical data, however, are considered in their approach only as a means for implementing a monitor. Only the object domains may vary in the course of time.

In [Ari91], and [Rod91] an informal introduction to the field of evolving information systems is provided, although the term evolving information systems is not used, but rather *schema evolution*. Within the class of evolving information systems, extensions of *object-oriented modelling techniques* with a time dimension (both on instance and type level) can be seen as a first subclass, providing first results on a way of supporting for evolving information systems based on the object-oriented paradigm. A taxonomy for type evolution in object-oriented databases is provided, in [SZ86]. The ORION project ([BKKK87], [KBC⁺89]) offers a more detailed taxonomy, together with a (semi-formal) semantics of schema updates restricted to object-oriented databases. The ORION system, together with the GemStone ([PS87], [BMO⁺89]) and Sherpa ([NR89]) systems, are among the first object-oriented database systems to support schema/type evolution. In the Cocoon project ([Tre91], [TS92]) an approach to the evolution of schemata in object-oriented databases is followed in which schema objects (e.g. object types) are considered to be objects like others (from the application). In this thesis a similar approach is adopted, in which objects of both levels of abstraction are considered to be objects describing an evolution over the course of time. This thesis differs from the Cocoon project in that it focuses on a conceptual way of modelling and communicating, rather than the implementation of a system supporting schema evolution.

A second subclass of evolving information systems, providing first results on a way of modelling, can be found in the field of version modelling ([Kat90], [MBJK90], [JMSV92], [Lip92]). One can argue that an evolving information system is a version management system, in which the application model is the object to be versioned. An important requirement for evolving information systems, not covered by version modelling systems, is that changes to the structure can be made on-line. In version modelling, a structural change still requires the replacement of the old system by a new system, and a costly conversion of the old population into a new population conforming to the new schema, further it is not possible to provide well-formedness rules with regards to the evolution of application models.

A third subclass of research on evolving information systems extends a manipulation language for relation models with historical operations, both on population and schema level leading to a way of communicating for evolving information systems. An example of this approach can be found in [MS90a], in which an algebra is presented that allows relational tables to evolve by changing their arity. This direction is similar to the ORION project ([BKKK87], [KBC⁺89]), in that a manipulation language is extended by operations supporting schema evolution. It is not clear, however, if these languages are able to provide an adequate disclosure that will support the users in finding their way in all versions of the underlying information structure, and the populations conforming to these information structures. Ideally, this disclosure is supported by mechanisms known from the information retrieval world such as *query by navigation* ([BW90d], [BW92b]) allowing for an explorative search through all the stored information.

Finally, two recent articles ([PL93], [EO93]) provide some first indications for a way of working with respect to the evolution of conceptual schemas.

1.6 Thesis Outline

In this thesis, we first develop a general (modelling technique independent) theory for evolving information systems. Then we apply this theory to existing modelling techniques, thus providing a quality check on the general theory. The structure of the main body of this thesis is therefore:

General theory

A general theory for the evolution of application models is provided in chapter 2 and 3. This theory is set up in such a way that a concrete modelling technique can be seen as a parameter, and as a result, this theory is reasonably modelling technique independent. In chapter 4, the focus is on recording and correction of information in an evolving information system. We provide a taxonomy of updates, that results in a three level architecture distinguishing between an event level, recording level and correction level for updates.

Applying the theory

The general evolution theory is applied to the PSM data modelling technique, resulting in EVORM, a data modelling technique supporting evolution of data models, in chapter 5. A proper way of communicating for evolving application models, the Elisa-D language, is introduced in chapter 6. The integrated PSM–LISA–D–Hydra framework is extended with the notion of evolution in chapter 7.

The requirement demanding an appropriate disclosure mechanism for evolving information systems is addressed in chapter 8.

In chapter 9, the requirements on evolving information systems, and an associated way of modelling and communicating, are revisited and the contents of this thesis are evaluated with respect to these requirements. Finally, also a research agenda for evolving information systems is provided. This thesis may have just covered the tip of the research iceberg in the field of evolving information systems.

Remark 1.6.1

It is advisable to read the appendixes first, these define the mathematical notations used in this thesis, and provide an overview of the graphical symbols used in PSM. □

Chapter 2

Information Structure Universe

The Universe - some information to help live in it.

1 Area: Infinite

The Hitch Hiker's Guide to the Galaxy offers this definition of the word 'Infinite'.

Infinite: Bigger than the biggest thing ever and then some. Much bigger than that in fact, really amazingly immense, a totally stunning size, real 'wow, that's big', time. Infinity is just so big that by comparison, bigness itself looks really tiny. Gigantic multiplied by colossal multiplied by staggeringly huge is the sort of concept we're trying to get across here.

...

...

4 Population: None

It is known that there are an infinite number of worlds, simply because there is an infinite amount of space for them to be in. However, not every one of them is inhabited. Therefore, there must be a finite number of inhabited worlds. Any finite number divided by infinity is as near to nothing as makes no odds, so the average population of all the planets in the Universe can be said to be zero. From this it follows that the population of the whole Universe is also zero, and that any people you may meet from time to time are merely the products of a deranged imagination.

From: "The Restaurant at the End of the Universe",
Douglas Adams, Pan Books Ltd.

2.1 Introduction

It is intended that an evolving information system should be able to handle updates of all components of application models. Such an application model is specified conforming to a set of (complementary) modelling techniques. These modelling techniques together provide a way of modelling for (states of) evolving information systems. When defining such a way of modelling, adequate modelling techniques must be selected for the components of the application model, and extended using the notion of evolution.

Due to the existence of the Methodology Jungle, it is not considered wise to simply choose *one* particular modelling technique for each component of an application model. In this, and the next chapter, which are entirely based on [PW93], [PW95a], and [PW94], a general theory for the evolution of application models is provided. This theory tries to make only weak assumptions on the underlying modelling techniques. The outcome of this exercise will be a theory for the way of modelling of evolving information systems, which can be applied to a wide range of modelling

techniques. The result of such an application is a modelling technique (for a part of the application model) supporting evolution.

In this chapter, the focus is on (generalised) information structures. In our view, the core of an application model is formed by the information structure, as all other components of the application model refer to (are formulated in terms of) this structure. For this purpose we introduce the notion of information structure universe, abstracting from a concrete data modelling technique and describing the minimal requirements for such a technique. Consequently, the results of this chapter are applicable to a wide range of data modelling techniques, including ER ([Che76]), EER ([HNSE87]), INFOMOD ([JG87]), IFO ([AH87]), NIAM ([NH89]), FORM ([HO92]) and PSM ([HW93]). Chapter 5 provides a major “return on investment” for the general theory, in the form of “theorems for free”.

The main issues in this chapter are *object typing*, *type relatedness*, *filtering of inheritance relations* and *identification of objects*. Populations of information structures are discussed in chapter 3, since populations cannot be discussed properly without taking their evolution into consideration. The following chapter, also addresses the well-formedness of evolution of information structures.

2.2 Generalised Information Structures

The kernel of the theory for evolving application models is formed by the *information structure universe*, which fixes the evolution space for information structures. Thus, the information structure of the application model being considered, is an element of this information structure universe.

The information structure universe, for a given modelling technique, is determined by a set \mathcal{O} of object types, together with relations \sim and \rightsquigarrow . The actual universe then is formed by taking all subsets of \mathcal{O} , however, not all sets of object types correspond to a correct information structure, therefore the criterion IsSch (is schema) is introduced to restrict the set of possible information structures.

An information structure version, at a point in time t , is identified by a set of object types $\mathcal{O}_t \subseteq \mathcal{O}$, where \sim and \rightsquigarrow specify the structure within the version. An information structure universe is identified by its components, this leads to the following definition:

Definition 2.2.1

An information structure universe (for a particular modelling technique) is determined by a structure:

$$\mathcal{U}_{\text{IS}} \triangleq \langle \mathcal{L}, \mathcal{N}, \sim, \rightsquigarrow, \text{IsSch} \rangle$$

where

1. \mathcal{L} are label types,
2. \mathcal{N} are the non-label types,
3. $\mathcal{O} \triangleq \mathcal{L} \cup \mathcal{N}$ forms the set of all object types such that the two groups of object types are disjoint: $\mathcal{L} \cap \mathcal{N} = \emptyset$,
4. \sim and \rightsquigarrow are binary relations over \mathcal{O} and
5. IsSch is a unary relation over \mathcal{O} .

□

The components of the information structure universe are explained in the remainder of this section, and furthermore, the definition is refined by the introduction of a set of axioms (ISU) for information structure universes. The evolution of an information structure is limited to the universe defined above.

Further refinements of the information structure universe will depend on the data modelling technique chosen. In chapter 5 we provide such an elaboration for the EVORM case. In the general theory, an information structure universe is assumed to provide (at least) the above components, which are available in all conventional high-level data modelling techniques. These components, and their properties, are discussed in the remainder of this chapter.

2.2.1 Object types

The center of an information structure universe is formed by the set of object types (referred to as object classes in object-oriented approaches). A distinction is made in data modelling between objects that can be represented directly and objects that cannot be represented directly ([HPW93]). In ER, this distinction is reflected by the difference between entity types and the value types associated to attribute types, while in NIAM and PSM this distinction corresponds to the difference between entity types and label types. Labels can be represented directly on a communication medium, while entities depend on labels for their representation. Label types are also called *concrete* object types, as opposed to entity types which are referred to as *abstract* object types. As a consequence, two major groups of object types are distinguished. Label types (\mathcal{L}) form the first group, and abstract object types form the class of non-label types (\mathcal{N}).

The set of all object types is defined as: $\mathcal{O} \triangleq \mathcal{L} \cup \mathcal{N}$. For an information structure version \mathcal{O}_t , the set of actual label types and non-label types is then: $\mathcal{L}_t \triangleq \mathcal{L} \cap \mathcal{O}_t$ and $\mathcal{N}_t \triangleq \mathcal{N} \cap \mathcal{O}_t$. The example shown in figure 1.13 (page 15) contains the following object types:

nine non-label types: Record, Song, Frequency, Recording, Lending-frequency, Title, Artist, Author and Times,
and four label types, being the value types associated to the attribute types: Title, Artist, Author and Times.

Remark 2.2.1

In most ER variants, at least three classes of non-label types are present. In the example in figure 1.13 (page 15) we have:

*three entity types: Record, Song and Frequency,
two relationship types: Recording and Lending-frequency,
and four attribute types: Title, Artist, Author and Times.*

□

2.2.2 Type relatedness

The relation $\sim \subseteq \mathcal{O} \times \mathcal{O}$ expresses *type relatedness* between object types (see [HW93]). Object types x and y are termed *type related* ($x \sim y$) if populations of object types x and y can have values in common in any version of the application model. For the data model depicted in figure 1.13 (page 15), the type relatedness relation is the identity relation: $x \sim x$ for all object types x . Type relatedness corresponds to mode equivalence in programming languages ([WMP⁺76]). Typically, subtyping and generalisation lead to type related object types.

The symmetry and reflexivity of this relation is enforced by the following axioms:

[ISU1] (*reflexive*) $x \sim x$

[ISU2] (*symmetric*) $x \sim y \Rightarrow y \sim x$

The separation of the concrete and abstract worlds, has the following consequence for type relatedness:

[ISU3] (*separation*) $x \sim y \Rightarrow x, y \in \mathcal{L} \vee x, y \in \mathcal{N}$

An example of a more complex type relatedness relation is provided in the PSM data model given in figure 2.1. In this example, A, B, C, D are object types, and the solid arrow stands for a subtyping (specialisation) relation, the dotted arrows represent generalisations. A major difference between generalisation and specialisation is that the population of (specialised) subtypes is defined by a subtype defining rule in terms of the population of the supertype, whereas the population of a generalised object type is the union of the populations of the underlying specifiers ([AH87],[HW93]). The type relatedness relation for the data model of figure 2.1 is therefore: $A \sim D, A \sim C, B \sim D, B \sim C, D \sim C$ (ignoring the symmetry and reflexivity of \sim).

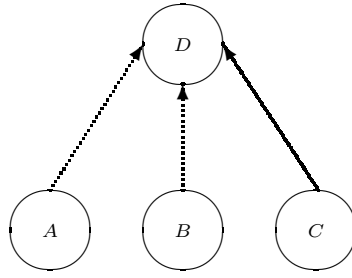


Figure 2.1: A data model with generalisation and specialisation

2.2.3 The identification hierarchy

The notion of object *identification* plays a crucial role in data modelling. In a proper information structure, each object type should be identifiable in terms of label types. If some non-label object type is not identifiable, its (abstract) instances cannot be denoted in terms of concrete label instances. This is considered to be an undesirable situation. The identifiability of object types provides proof that the object types are populatable ([Hof93]).

Since the general theory does not pose any assumptions on the underlying structure of object types, we will consider identification issues in the context of generalisation and specialisation. By underlying structure, we mean the construction of new object types in terms of other object types, for instance, the underlying structure of a fact type is formed by the set of object types playing a *role* in the fact type.

In a subtype hierarchy, a subtype inherits its identification from its supertype, in a generalisation hierarchy the identification of a generalised object type is inherited from its specifiers. Thus for the data model depicted in figure 2.1 this means that instances of *C* are identified in the same way as instances of *D*. The identification of instances of *D* depends on the identification of instances of *A* or *B* (note that an instance of *D* is either an instance of *A* or an instance from *B*). In the data model depicted in figure 1.14 (page 16), the instances of *Record* and *CD* are identified in the same way as instances of *Medium*.

An object type from which the identification is inherited, is termed an *ancestor* of that object type. The identification hierarchy is provided by the relation $x \rightsquigarrow y$, meaning x is an ancestor of y . For figure 1.14 (page 16) this leads to: $\text{Medium} \rightsquigarrow \text{Record}$ and $\text{Medium} \rightsquigarrow \text{CD}$. The identification hierarchy is both transitive and irreflexive.

[ISU4] (*irreflexive*) $\neg x \rightsquigarrow x$

[ISU5] (*transitivity*) $x \rightsquigarrow y \rightsquigarrow z \Rightarrow x \rightsquigarrow z$

Similar axioms can be found as properties in the literature on typing theory for databases (see for instance [BW90c], [Oho90] and [CW85]). The difference, between these properties and the above axioms is that we do not make any assumption about the underlying structure of object types, i.e. we abstract from these underlying structures. Due to the abstraction of the underlying structures, the inheritance of properties cannot be derived from these underlying structures. As a consequence, such properties must be stated explicitly as axioms.

Object types without ancestor, are called *roots*: $\text{IsRoot}(x) \triangleq \neg \exists z [z \rightsquigarrow x]$. We employ $x \rightsquigarrow y$ as an abbreviation for $x = y \vee x \rightsquigarrow y$. The roots of an object type y are now found by:

$$x \text{ RootOf } y \triangleq \text{IsRoot}(x) \wedge x \rightsquigarrow y$$

This relation is reflexive for root object types:

Proposition 2.2.1 (*root reflexivity*) $x \text{ RootOf } y \Rightarrow x \text{ RootOf } x$

Next we focus on direct ancestors of object types within the identification hierarchy. We call p a direct ancestor (a parent) of x , denoted as $p \text{ ParentOf } x$, if:

$$p \rightsquigarrow x \wedge \neg \exists_z [p \rightsquigarrow z \rightsquigarrow x]$$

The existence of direct ancestors is postulated by:

[ISU6] (*direct ancestors*) $a \rightsquigarrow x \Rightarrow \exists_p [a \rightsquigarrow p \wedge p \text{ ParentOf } x]$

The (complete) identification of a non-root object type is derived from the identification properties of its ancestors. Thus, the identification of a non-root object type can only be complete if all its ancestors are. This finite boundedness of the identification hierarchy is expressed by the following schema of induction:

[ISU7] (*parent induction*) If $\forall_{x \in \mathcal{O}} [\forall_{p:p \text{ ParentOf } x} [F(p)] \Rightarrow F(x)]$, then $\forall_{x \in \mathcal{O}} [F(x)]$.

Note that this axiom is only meaningful because of axiom ISU4. If $x \rightsquigarrow x$, any property $P(x)$ would become provable by the ancestor induction axiom.

Remark 2.2.2

This schema of induction does not have an explicit base step. One might expect the induction schema to be:

If $\text{IsRoot}(y) \Rightarrow F(y)$ (base step)
and $\forall_{y \in \mathcal{O}} [\forall_{x:x \rightsquigarrow y} [F(x)] \Rightarrow F(y)]$, (induction step)
then $\forall_x [F(x)]$.

The $\text{IsRoot}(y)$ base step is (see the definition of IsRoot above) already included in

$$\forall_{y \in \mathcal{O}} [\forall_{x:x \rightsquigarrow y} [F(x)] \Rightarrow F(y)]$$

□

The parent induction schema leads to the following, more convenient induction schema:

Theorem 2.2.1 (*ancestor induction*) If $\forall_{x \in \mathcal{O}} [\forall_{a:a \rightsquigarrow x} [F(a)] \Rightarrow F(x)]$, then $\forall_{x \in \mathcal{O}} [F(x)]$.

Proof:

If F has property $\forall_{x \in \mathcal{O}} [\forall_{a:a \rightsquigarrow x} [F(a)] \Rightarrow F(x)]$. Now consider $G(x) = \forall_{a:a \rightsquigarrow x} [F(a)]$. Using parent induction, we prove the stronger property $\forall_{x \in \mathcal{O}} [G(x)]$, which directly implies $\forall_{x \in \mathcal{O}} [F(x)]$.

Induction hypothesis: $\forall_{p:p \text{ ParentOf } x} [G(p)]$ for any x .

Due to axiom ISU6 we have: $\forall_{a:a \rightsquigarrow x} \exists_p [a \rightsquigarrow p \wedge p \text{ ParentOf } x]$

From this we derive:

$$\begin{aligned} \forall_{a:a \rightsquigarrow x} \exists_p [a \rightsquigarrow p \wedge p \text{ ParentOf } x] &\Rightarrow \{\text{induction hypothesis}\} \\ \forall_{a:a \rightsquigarrow x} \exists_p [a \rightsquigarrow p \wedge G(p)] &\Rightarrow \{\text{definition of } G\} \\ \forall_{a:a \rightsquigarrow x} [F(a)] &\Rightarrow \{\text{assumption}\} \\ \forall_{a:a \rightsquigarrow x} [F(a)] \wedge F(x) &\equiv \{\text{elaborate}\} \\ \forall_{a:a \rightsquigarrow x} [F(a)] &\equiv \{\text{definition of } G\} \\ G(x) & \end{aligned}$$

□

For every data model expressed in a conventional data modelling technique, an ancestor and root relation can be derived. If no specialisations or generalisations are present in a particular data model, the associated ancestor relation is empty, and as a result, the root relation is the identity relation. For instance the root relation for figure 1.13 (page 15) is: $x \text{ RootOf } x$ for every object type x . When the data model under consideration contains specialisations or generalisations, the relations \leadsto and RootOf will be less trivial. The relation RootOf (*root dependency graph*) of the data model in figure 2.1 is shown in figure 2.2.

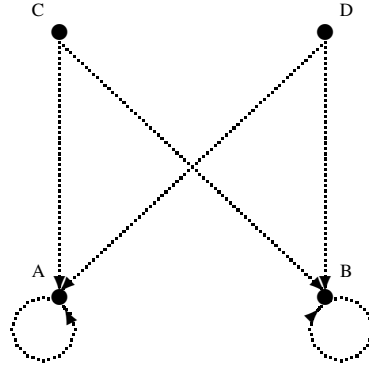


Figure 2.2: Root dependency graph

2.2.4 Inheritance of type relatedness

In this section we introduce two special types of properties with respect to inheritance, for which some general theorems will be proven in a later section. A property P is *preserved* by a relation R , if for all x, y :

$$P(x) \wedge R(x, y) \Rightarrow P(y)$$

Furthermore, a property P is *reflected* by a relation R , if for all x :

$$P(x) \wedge \exists_a [R(a, x)] \Rightarrow \exists_a [P(a) \wedge R(a, x)]$$

If a property P is reflected and preserved by a relation R , then P is said to be *filled* by relation R .

A first example of a filled property, is type relatedness. From the ancestor relation it follows intuitively that object types may have instances in common with their ancestors. This implies that object types not only inherit identification from their ancestors, but also type relatedness. Let the relation P_x be defined by $P_x(y) \triangleq x \sim y$ for all x , then the inheritance of type relatedness is enforced by the following axiom:

[ISU8] (*inheritance and foundation of type relatedness*) The relation P_x is filled by \leadsto , for all $x \in \mathcal{O}$.

Some immediate consequences are:

Corollary 2.2.1

1. $x \leadsto y \Rightarrow x \sim y$
2. $x \leadsto y \Rightarrow x, y \in \mathcal{L} \vee x, y \in \mathcal{N}$
3. $x \text{ RootOf } y \Rightarrow x \sim y$

Some examples of properties which are also filled by \leadsto are:

1. *true*.
2. *false*.
3. The relation Q_r , defined by $Q_r(x) \triangleq r \text{ RootOf } x$.

2.2.5 Correctness of information structures

An information structure is spanned by a set of object types. Not all sets of object types taken from \mathcal{O} correspond to a (syntactically) correct information structure. Therefore, a technique dependent predicate $\text{IsSch} \subseteq \wp(\mathcal{O})$ needs to be supplied that designates which sets of object types form a correct information structure.

Figure 2.3, gives a typical example of an incorrect information structure. Fact type f is an objectified fact type, consisting of only one role p . This role has as its base (the player of the role) fact type f . So fact type f is simply referring to itself. Therefore, fact type f is not (finitely) populatable, and therefore the schema should be rejected.

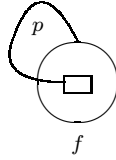


Figure 2.3: Incorrect information structure

2.2.6 An example: ER

We will take Chen's ([Che76]) ER model (extended with subtyping) as an example of an information structure universe. In this case, the information structure universe is:

Label Types The set of label types \mathcal{L} in ER corresponds to the value types associated to the attribute types.

Non-Label Types The set of non-label types \mathcal{N} is defined as the set of relationship types and entity types.

Inheritance ER only contains the notion of subtyping. So for each subtype x of a supertype y we have: $y \rightsquigarrow x$. The complete inheritance relation \rightsquigarrow is then obtained by application of the transitive closure.

Type Relatedness Two subtypes of the same supertype are type related, furthermore, subtyping is the only way in ER to make type related object types. The resulting subtyping hierarchy always has a unique top element. Let $\sqcap(x)$ denote the unique top element of the subtyping hierarchy containing object type x . Type relatedness for ER can then be defined as: $x \sim y \triangleq \sqcap(x) = \sqcap(y)$.

A definition of the IsSch predicate is not provided here, as it would require a detailed formalisation of the ER modelling technique.

2.3 Properties of Information Structures

The axioms until now have attempted to model the concepts of type relatedness, object type and inheritance. In this section, some useful properties of information structure universes are derived that also illustrate the validity of the ISU axioms. In chapter 5, these properties will turn out to be 'theorems for free', with respect to the EVORM case.

The first general theorem is concerned with inheritance of properties, and states that preservation of a property implies the validity of the property for all descendants:

Theorem 2.3.1 (*inheritance schema*) If property P is preserved by \rightsquigarrow , then it is also preserved by RootOf :

$$P(x) \wedge x \text{ RootOf } y \Rightarrow P(y)$$

Proof:

If P is preserved by \rightsquigarrow , then:

$$P(x) \wedge x \text{ RootOf } y \quad \Rightarrow \{ \text{definition of RootOf} \}$$

$$P(x) \wedge x \rightsquigarrow y \quad \Rightarrow \{ P \text{ is preserved by } \rightsquigarrow \}$$

$$P(y)$$

□

The second general theorem is concerned with the foundation of properties. Properties of object types can be traced back to their roots:

Theorem 2.3.2 (*basic foundation schema*) If property P is reflected by \rightsquigarrow , then it is also reflected by RootOf:

$$P(y) \wedge \neg \text{IsRoot}(y) \Rightarrow \exists_x [P(x) \wedge x \text{ RootOf } y]$$

Proof:

If P is reflected by \rightsquigarrow . We apply ancestor induction to prove the theorem. From the induction hypothesis follows:

$$\forall_{v \rightsquigarrow w} [P(v) \wedge \neg \text{IsRoot}(v) \Rightarrow \exists_z [P(z) \wedge z \text{ RootOf } v]]$$

As $\text{IsRoot}(v) \Rightarrow v \text{ RootOf } v$, we can reformulate this to:

$$\forall_{v: v \rightsquigarrow w} [P(v) \Rightarrow \exists_z [P(z) \wedge z \text{ RootOf } v]]$$

From this we prove:

$$P(w) \wedge \neg \text{IsRoot}(w) \Rightarrow \exists_z [P(z) \wedge z \text{ RootOf } w]$$

If $P(w) \wedge \neg \text{IsRoot}(w)$, then we have:

$$P(w) \wedge \neg \text{IsRoot}(w) \quad \Rightarrow \{ P \text{ is reflected by } \rightsquigarrow \}$$

$$\exists_u [u \rightsquigarrow w \wedge P(u)] \quad \Rightarrow \{ \text{induction hypothesis} \}$$

$$\exists_u [(P(u) \Rightarrow \exists_z [P(z) \wedge z \text{ RootOf } u]) \wedge u \rightsquigarrow w \wedge P(u)] \quad \Rightarrow \{ \text{modus ponens} \}$$

$$\exists_u [\exists_z [P(z) \wedge z \text{ RootOf } u] \wedge u \rightsquigarrow w] \quad \Rightarrow \{ \text{definition of RootOf} \}$$

$$\exists_z [P(z) \wedge z \text{ RootOf } w]$$

□

From this theorem, and the observation that $y \text{ RootOf } y$ if $\text{IsRoot}(y)$, immediately follows the following more convenient formulation of the (base) foundation schema:

Corollary 2.3.1 (*foundation schema*) If property P is reflected by \rightsquigarrow , then:

$$P(y) \Rightarrow \exists_x [P(x) \wedge x \text{ RootOf } y]$$

As stated before, a property P which is both reflected and preserved by a relation R , is said to be filled by the relation R . For properties filled by \rightsquigarrow , the inheritance can be traced from parents:

Theorem 2.3.3 (*parental foundation*) If property P is filled by \rightsquigarrow , then P is reflected by ParentOf:

$$P(x) \wedge \neg \text{IsRoot}(x) \Rightarrow \exists_p [P(p) \wedge p \text{ ParentOf } x]$$

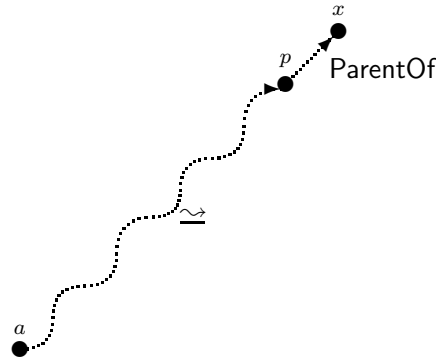


Figure 2.4: Existing ancestors

Proof:

Let P be filled by \leadsto , then:

$$\begin{aligned}
 P(x) \wedge \neg \text{IsRoot}(x) &\Rightarrow \{P \text{ is reflected by } \leadsto\} \\
 \exists_z [P(z) \wedge z \leadsto x] &\Rightarrow \{\text{axiom ISU6 (page 25), see also figure 2.4}\} \\
 \exists_{p,z} [P(z) \wedge z \leadsto p \wedge p \text{ ParentOf } x] &\Rightarrow \{P \text{ is preserved by } \leadsto\} \\
 \exists_{p,z} [z \leadsto p \wedge P(p) \wedge p \text{ ParentOf } x] &\Rightarrow \{\text{simplify}\} \\
 \exists_p [P(p) \wedge p \text{ ParentOf } x] &
 \end{aligned}$$

□

We discuss some special inheritance properties filled by RootOf, in the remainder of this section.

2.3.1 Inheritance of type relatedness

The first group of inheritance properties under consideration are the P_x relations. These relations, are defined by $P_x(y) \triangleq x \sim y$ for all x . Application of theorem 2.3.1, corollary 2.3.1 and theorem 2.3.3 respectively to relation P_a , for some a , yields:

Corollary 2.3.2 Relation \sim is preserved by RootOf:

$$a \sim x \wedge x \text{ RootOf } y \Rightarrow a \sim y$$

Corollary 2.3.3 Relation \sim is reflected by RootOf, which can be formulated more strongly as:

$$a \sim y \Rightarrow \exists_x [a \sim x \wedge x \text{ RootOf } y]$$

Corollary 2.3.4 Relation \sim is reflected by ParentOf:

$$a \sim x \wedge \neg \text{IsRoot}(x) \Rightarrow \exists_p [a \sim p \wedge p \text{ ParentOf } x]$$

If two object types are type related, then they may share instances. Using the above results, this implies that if two object types share a root, they should be type related. This is formulated in the following lemma:

Lemma 2.3.1 $v \text{ RootOf } x \wedge v \text{ RootOf } y \Rightarrow x \sim y$

Proof:

$$v \text{RootOf } x \wedge v \text{RootOf } y \quad \Rightarrow \{ \text{corollary 2.2.1 (page 26)} \}$$

$$x \sim v \wedge v \text{RootOf } y \quad \Rightarrow \{ \text{corollary 2.3.2 (page 29)} \}$$

$$x \sim y$$

□

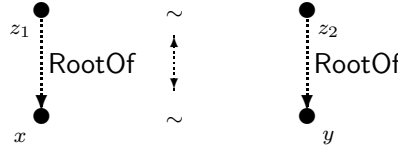


Figure 2.5: Propagation of Type Relatedness

The following theorem, illustrated in figure 2.5, shows that type relatedness of object types is equivalent to type relatedness of their roots:

Theorem 2.3.4 (*type relatedness propagation*)

$$x \sim y \quad \Longleftrightarrow \quad \exists_{z_1, z_2} [z_1 \sim z_2 \wedge z_1 \text{RootOf } x \wedge z_2 \text{RootOf } y]$$

Proof:

We prove this theorem by applying the combination of corollary 2.3.2 (page 29) and 2.3.3 twice:

$$\exists_{z_1, z_2} [z_1 \sim z_2 \wedge z_1 \text{RootOf } x \wedge z_2 \text{RootOf } y] \quad \equiv \{ \text{apply corollary 2.3.2 and 2.3.3 to } z_1 \}$$

$$\exists_{z_2} [x \sim z_2 \wedge z_2 \text{RootOf } y] \quad \equiv \{ \text{apply corollary 2.3.2 and 2.3.3 to } z_2 \}$$

$$x \sim y$$

□

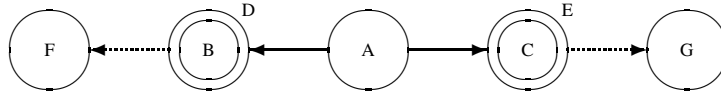


Figure 2.6: Data model with propagation of type relatedness

As an illustration of this theorem, consider the PSM data model in figure 2.6. It contains two generalisations, two specialisations, and two power types (D , E). *Power types* are the data modelling pendant of power sets used in set theory. The instances of object types D and E are sets of instances of B and C respectively. The RootOf relation for this data model, is given in figure 2.7. The type relatedness of D and E , which itself follows from the type relatedness of B and C ([HW93]), is propagated to F and G by the RootOf relationship and theorem 2.3.4. The inheritance of type relatedness via type constructions, e.g. power typing, is elaborated in [HW93], [HPW93].

The above theorem allows on its turn for the formulation of the following theorem, expressing intuitively that if an object type shares all its roots with another object type, it also shares all its type related object types.

Theorem 2.3.5

$$x \sim y \wedge \forall_r [r \text{RootOf } y \Rightarrow r \text{RootOf } z] \Rightarrow x \sim z$$

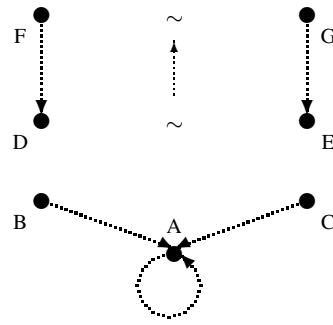


Figure 2.7: Root dependency graph showing propagation of type relatedness

Proof:

$$\begin{aligned}
 x \sim y \wedge \forall_r [r \text{ RootOf } y \Rightarrow r \text{ RootOf } z] &\Rightarrow \{\text{corollary 2.3.3 (page 29)}\} \\
 \exists_w [x \sim w \wedge w \text{ RootOf } y] \wedge \forall_r [r \text{ RootOf } y \Rightarrow r \text{ RootOf } z] &\Rightarrow \{\text{modus ponens}\} \\
 \exists_w [x \sim w \wedge w \text{ RootOf } z] &\Rightarrow \{\text{axiom ISU8 (page 26)}\} \\
 x \sim z &
 \end{aligned}$$

Figure 2.8 illustrates this proof. Note that z and x may have more roots than y . □

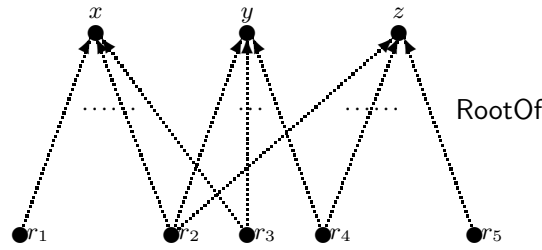


Figure 2.8: Shared roots

2.3.2 Inheritance of truth

Next we consider the property *true* over \mathcal{O} . In this case, theorem 2.3.1 leads to an obvious statement. Application of corollary 2.3.1, however, yields:

Corollary 2.3.5 $\forall_{y \in \mathcal{O}} \exists_x [x \text{ RootOf } y]$

stating that every object type has a root.

2.3.3 Inheritance of roots

The property Q_r , which has been defined as $Q_r(x) \triangleq r \text{ RootOf } x$, is filled by \sim . For this property, theorem 2.3.1 and corollary 2.3.1 are trivial statements. From theorem 2.3.3 we get:

Corollary 2.3.6

$$r \text{ RootOf } x \wedge \neg \text{IsRoot}(x) \Rightarrow \exists_p [r \text{ RootOf } p \wedge p \text{ ParentOf } x]$$

stating that a direct ancestor of a root object type exists on a path in the identification hierarchy from a non-root object type to its root,

2.4 Filtering a Hierarchy

When applying the general evolution theory to a concrete modelling technique, the identification hierarchy needs to be split into subhierarchies. For instance, PSM distinguishes two ‘flavours’ of inheritance in the identification hierarchy: generalisation and specialisation. This section provides a filtering mechanism to facilitate the dissection of the identification hierarchy into hierarchies dealing with one ‘flavour’ of inheritance, resulting in a palette of ‘flavours’ each of which covers a part of the original identification hierarchy. Axioms are provided that guarantee the completeness of this ‘flavour palette’ with respect to the original identification hierarchy.

Generally, we call a binary relation $R \subseteq \mathcal{O} \times \mathcal{O}$ a *filter relation* (an inheritance flavour) on the identification hierarchy \rightsquigarrow if it adheres to the following rules:

[F1] (*transitivity completeness*) If $x \rightsquigarrow y \rightsquigarrow z$, then:

$$R(x, y) \wedge R(y, z) \iff R(x, z)$$

[F2] (*choice completeness*) If $p \text{ ParentOf } x$ and $q \text{ ParentOf } x$, then:

$$R(p, x) \Rightarrow R(q, x)$$

The filtered identification hierarchy \rightsquigarrow_R then is defined by:

$$x \rightsquigarrow_R y \triangleq R(x, y) \wedge x \rightsquigarrow y$$

In this case we will speak of R -ancestors rather than ancestors. As before, $x \rightsquigarrow_R y$ is used as a shorthand for $x = y \vee x \rightsquigarrow_R y$. We call p a direct R -ancestor (an R -parent) of x , denoted as $p \text{ ParentOf}_R x$, if:

$$p \rightsquigarrow_R x \wedge \neg \exists_a [p \rightsquigarrow_R a \rightsquigarrow_R x]$$

In the resulting subhierarchy, object types may have no R -ancestors:

$$\text{IsRoot}_R(x) \triangleq \neg \exists_z [z \rightsquigarrow_R x]$$

Such object types are denoted as R -roots, and are found by:

$$x \text{ RootOf}_R y \triangleq x \rightsquigarrow_R y \wedge \text{IsRoot}_R(x)$$

The first property on such filtered hierarchies states that the filtered hierarchy obeys the direct ancestorship (ParentOf) relation:

Lemma 2.4.1 $p \text{ ParentOf } x \wedge R(p, x) \iff p \text{ ParentOf}_R x$

Proof:

$$p \text{ ParentOf } x \wedge R(p, x) \equiv \{\text{definition of ParentOf}\}$$

$$p \rightsquigarrow x \wedge R(p, x) \wedge \neg \exists_z [p \rightsquigarrow z \rightsquigarrow x] \equiv \{\text{axiom F1}\}$$

From left to right this step is rather simple.

From right to left, however, the motivation is as follows:

if $p \rightsquigarrow z \rightsquigarrow x$ would hold for a certain z , then

as $R(p, x)$ we also have: $R(p, z) \wedge R(z, x)$

leading to a contradiction.

Therefore, such a z does not exist.

$$p \rightsquigarrow x \wedge R(p, x) \wedge \neg \exists_z [p \rightsquigarrow z \rightsquigarrow x \wedge R(p, z) \wedge R(z, x)] \equiv \{\text{definition of } \rightsquigarrow_R\}$$

$$p \rightsquigarrow_R x \wedge \neg \exists_z [p \rightsquigarrow_R z \rightsquigarrow_R x] \equiv \{\text{definition of ParentOf}_R\}$$

$$p \text{ ParentOf}_R x$$

□

A direct consequence is the following corollary, which states that if an object type has an R -parent, all its parents are R -parents.

Corollary 2.4.1 $p \text{ ParentOf}_R x \wedge q \text{ ParentOf } x \Rightarrow q \text{ ParentOf}_R x$

Corollary 2.4.2 $\forall_{p:p \text{ ParentOf } x} [F(p)] \Rightarrow \forall_{q:q \text{ ParentOf}_R x} [F(q)]$

Filtering an inheritance relation leads to a proper information structure universe:

Theorem 2.4.1 If R is a filter relation, then $\langle \mathcal{L}, \mathcal{N}, \sim, \rightsquigarrow_R, \text{IsSch} \rangle$ is an information structure universe.

Proof:

We will prove \rightsquigarrow_R versions of the ISU axioms. The validity of axiom ISU1 $_R$, axiom ISU2 $_R$, axiom ISU3 $_R$ and axiom ISU4 $_R$ is obvious, while axiom ISU5 $_R$ is a direct consequence of axiom F1.

The remaining axioms:

axiom ISU6 $_R$

$$a \rightsquigarrow_R x \equiv \{\text{definition of } \rightsquigarrow_R\}$$

$$a \rightsquigarrow x \wedge R(a, x) \Rightarrow \{\text{axiom ISU6 (page 25)}\}$$

$$\exists_q [a \rightsquigarrow q \wedge q \text{ ParentOf } x] \wedge R(a, x) \equiv \{\text{axiom F1, definition of ParentOf}\}$$

$$\exists_q [a \rightsquigarrow_R q \wedge R(q, x) \wedge q \text{ ParentOf } x] \equiv \{\text{lemma 2.4.1 (page 32)}\}$$

$$\exists_q [a \rightsquigarrow_R q \wedge q \text{ ParentOf}_R x]$$

axiom ISU7 $_R$

Let F be a property such that $\forall_{a:a \text{ ParentOf}_R x} [F(a)] \Rightarrow F(x)$.

Suppose further that $\forall_{q:q \text{ ParentOf } x} [F(q)]$.

Then from corollary 2.4.2 (page 33) it follows that $\forall_{q:q \text{ ParentOf}_R x} [F(q)]$.

From the first assumption then follows $F(x)$.

As a result: $\forall_{q:q} \text{ParentOf}_x [F(q)] \Rightarrow F(x)$.

From axiom ISU7 (page 25) follows: $\forall_{x:x \in \mathcal{O}} [F(x)]$.

axiom ISU8_R

The preservation of P_x by \leadsto_R , directly follows from the preservation of P_x by \leadsto , and the observation:
 $x \leadsto_R y \Rightarrow x \leadsto y$.

The reflection follows from:

$$\begin{aligned}
 x \sim y \wedge \neg \text{IsRoot}_R(y) & \equiv \{\text{definition of IsRoot}_R\} \\
 x \sim y \wedge \neg \text{IsRoot}(y) \wedge \exists_a [a \leadsto_R y] & \Rightarrow \{\text{corollary 2.3.4 (page 29)}\} \\
 \exists_p [x \sim p \wedge p \text{ParentOf } y] \wedge \exists_a [a \leadsto_R y] & \Rightarrow \{\text{axiom ISU6}_R\} \\
 \exists_p [x \sim p \wedge p \text{ParentOf } y] \wedge \exists_q [q \text{ParentOf}_R y] & \Rightarrow \{\text{corollary 2.4.1 (page 33)}\} \\
 \exists_p [x \sim p \wedge p \text{ParentOf}_R y] & \Rightarrow \{\text{definition of ParentOf}_R\} \\
 \exists_p [x \sim p \wedge p \leadsto_R y] &
 \end{aligned}$$

□

As a direct result of the above theorem, all properties of \leadsto will, *a fortiori*, also hold for \leadsto_R . With axiom ISU6_R, corollary 2.4.1 can be strengthened to the following lemma, expressing the property that every object type which has an R -ancestor can only have R -parents:

Lemma 2.4.2 $\neg \text{IsRoot}_R(y) \wedge p \text{ParentOf } y \Rightarrow p \text{ParentOf}_R y$

Proof:

$$\begin{aligned}
 \neg \text{IsRoot}_R(x) \wedge p \text{ParentOf } x & \Rightarrow \{\text{definition of IsRoot}_R\} \\
 \exists_a [a \leadsto_R x] \wedge p \text{ParentOf } x & \Rightarrow \{\text{axiom ISU6}_R\} \\
 \exists_{a,q} [a \leadsto_R q \wedge q \text{ParentOf}_R x] \wedge p \text{ParentOf } x & \Rightarrow \{\text{corollary 2.4.1 (page 33)}\} \\
 p \text{ParentOf}_R x &
 \end{aligned}$$

□

Next, we focus on the relation between inheritance properties in an identification hierarchy, and a filtered version of such an hierarchy.

Theorem 2.4.2 If P is preserved by \leadsto , and R is a filter relation on \leadsto , then P is also preserved by \leadsto_R .

Proof:

If P is preserved by \leadsto

$$\begin{aligned}
 P(x) \wedge x \leadsto_R y & \Rightarrow \{\text{definition of } \leadsto_R\} \\
 P(x) \wedge x \leadsto y & \Rightarrow \{P \text{ is preserved by } \leadsto\} \\
 P(y) &
 \end{aligned}$$

□

If a relation P is reflected by \leadsto , then P does not have to be reflected by \leadsto_R . If $P(x)$, and x is not an R -root, then an ancestor a exists such that $P(a)$, as P is reflected by \leadsto . An R -ancestor a' also exists, however, we are not able to prove that a' has property P . This is illustrated in figure 2.9. Nevertheless, if P is also preserved by \leadsto , we can prove the following:

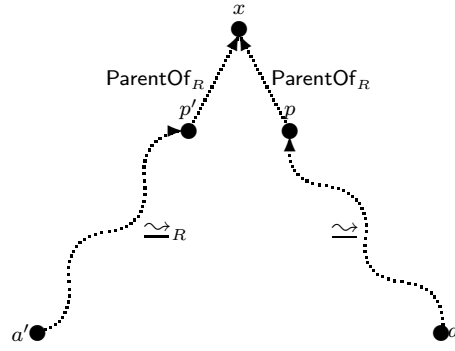


Figure 2.9: Filled inheritance property

Theorem 2.4.3 If P is filled by \leadsto , and R is a filter relation on \leadsto , then P is filled by \leadsto_R .

Proof:

If P is filled by \leadsto , and R a filter relation on \leadsto . From the previous lemma we know that P is preserved by \leadsto_R .

With respect to the reflection of P by \leadsto , if $P(y) \wedge \neg \text{IsRoot}_R(y)$, then we have:

$$\begin{aligned}
 P(y) \wedge \neg \text{IsRoot}_R(y) &\Rightarrow \{\text{definition of } \text{IsRoot}_R\} \\
 P(y) \wedge \neg \text{IsRoot}(y) \wedge \neg \text{IsRoot}_R(y) &\Rightarrow \{P \text{ is reflected by } \leadsto\} \\
 \exists_a [P(a) \wedge a \leadsto y] \wedge \neg \text{IsRoot}_R(y) &\Rightarrow \{\text{axiom ISU6 (page 25)}\} \\
 \exists_{a,p} [P(a) \wedge a \leadsto p \wedge p \text{ ParentOf } y] \wedge \neg \text{IsRoot}_R(y) &\Rightarrow \{P \text{ is preserved by } \leadsto\} \\
 \exists_p [P(p) \wedge p \text{ ParentOf } y] \wedge \neg \text{IsRoot}_R(y) &\Rightarrow \{\text{lemma 2.4.2 (page 34)}\} \\
 \exists_p [P(p) \wedge p \text{ ParentOf}_R y] &\Rightarrow \{\text{definition of } \leadsto_R\} \\
 \exists_p [P(p) \wedge p \leadsto_R y] &
 \end{aligned}$$

Proving that P reflects \leadsto_R , therefore P is also filled by \leadsto_R . □

The following property states that the existence of R -roots is bound by the original RootOf relation:

Lemma 2.4.3 $r \text{ RootOf } x \iff \exists_s [r \text{ RootOf } s \wedge s \text{ RootOf}_R x]$

Proof:

\implies Q_r is preserved by \leadsto , and therefore (theorem 2.4.2 (page 34)) is also preserved by \leadsto_R . Applying the foundation schema (corollary 2.3.1 (page 28)) for RootOf_R yields the result.

\impliedby From $r \text{ RootOf } s$ and $s \text{ RootOf}_R x$ follows: $\text{IsRoot}(r)$ and $r \leadsto s \leadsto x$, and therefore $r \text{ RootOf } x$. □

Filter relations R_1, \dots, R_n of \leadsto form an identification signature for the identification hierarchy \leadsto , if they span \leadsto :

Definition 2.4.1 (*identification signature*)

$$x \text{ ParentOf } y \Rightarrow \exists!_i [R_i(x, y)]$$

□

A direct result of this definition is:

Lemma 2.4.4 If R_1, \dots, R_n are an identification signature of \leadsto , then $\text{ParentOf}_{R_1}, \dots, \text{ParentOf}_{R_n}$ forms a partition of ParentOf .

Proof:

From the definition of an identification structure it immediately follows that if $i \neq j$, then ParentOf_{R_i} and ParentOf_{R_j} are disjunct. Further, we clearly have $\text{ParentOf}_{R_i} \subseteq \text{ParentOf}$, therefore:

$$\bigcup_{1 \leq i \leq n} \text{ParentOf}_{R_i} \subseteq \text{ParentOf}$$

Conversely, from the definition of an identification structure follows:

$$\text{ParentOf} \subseteq \bigcup_{1 \leq i \leq n} \text{ParentOf}_{R_i}$$

□

2.5 Conclusions

The notion of information structure universe has been defined, and properties of information structure versions in such universes have been proven in this chapter. The information structure universe serves as the core of the general evolution theory that is defined in the next chapter.

The information structure universe has been setup to be independent of any concrete data modelling technique, and is therefore applicable to a wide range of data modelling techniques. The properties of information structure versions, in particular the properties of filtered identification hierarchies, are revisited in chapter 5 where the general theory is applied to a concrete data modelling technique, here the properties turn out to be “theorems for free”, as the concrete data modelling technique provides a proper information structure universe.

Chapter 3

Evolution of Application Models

Time travel is increasingly regarded as a menace. History is being polluted.

The Encyclopaedia Galactica has much to say on the theory and practice of time travel, most of which is incomprehensible to anyone who hasn't spent at least four lifetimes studying advanced hypermathematics, and since it was impossible to do this before time travel was invented, there is a certain amount of confusion as to how the idea was arrived at in the first place. One rationalisation of this problem states that time travel was, by its very nature, discovered simultaneously at all periods of history, but this is clearly bunk.

From: "Life, the Universe and Everything",
Douglas Adams, Pan Books Ltd.

3.1 Introduction

In this chapter, we first describe the underlying way of thinking for the modelling of evolution used in this thesis. In the previous chapter, we provided a general way of modelling for information structures making only weak assumptions on the underlying method. On the basis of this way of modelling for information structures, a general way of modelling is defined for the evolution of application models. The resulting theory makes a distinction between the underlying information structure and its evolution on the one hand, and the description and semantics of operations on the information structure and its population on the other hand.

The way of modelling for information structures defined in the previous chapter can be applied to a wide range of data modelling techniques, due to its modelling technique independence. The description of the semantics of operations on the information structure and its population, is also provided in a technique independent fashion, yielding a way of modelling for evolution which can be applied to a wide range of action modelling techniques such as Task Structures ([WHO92]), DFD ([You89]) and ExSpect ([HSV89]). Further, due to the inheritance of properties in identification hierarchies, discussed in the previous chapter, the connection to object-oriented modelling methods ([SM88], [CY90], [RBP⁺91]) can easily be established.

In the next section, we start by discussing some approaches to modelling the evolution of application models, and define the approach taken in this thesis. The structure of the formal part of this chapter is best explained by figure 3.1. The previous chapter defined the notion of information structure universe, and provided us with a typing mechanism that is embodied in the ISU (Information Structure Universe) rules. The universe for application models is introduced in section 3.3, resulting in AMU (Application Model Universe) rules. These two classes of rules form the kernel of our axiomatic framework for the modelling of evolution. In section 3.4 we discuss rules for version management

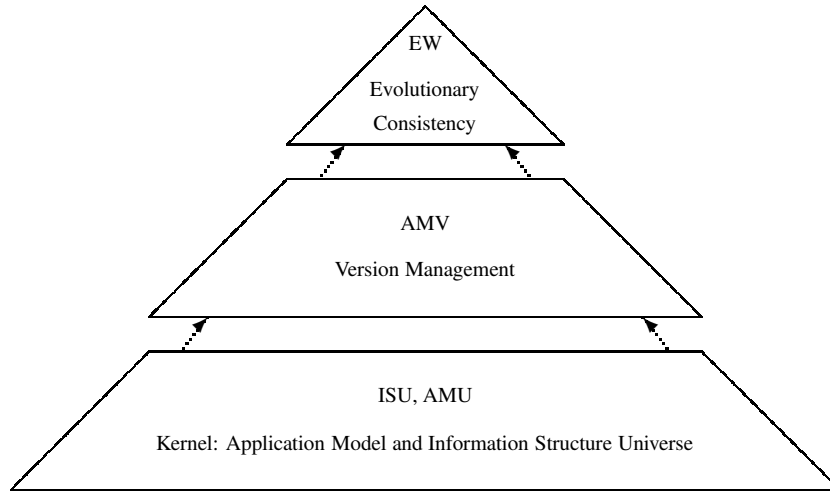


Figure 3.1: Axiomatic framework

of application models, leading to AMV (Application Model Version) rules describing well-formedness of versions. The version management rules, along with a time axis \mathcal{T}_s , serve in section 3.5 as the base for EW (Evolution Well-formedness) rules, describing what constitutes well-formed evolution of an information system. When applying the general evolution theory to a concrete (data) modelling technique, as is done in chapter 5, the modelling technique must provide a typing system that conforms to the typing axioms of the general theory, and well-formedness rules for versions of the models.

3.2 Modelling Evolution

This section addresses the modelling of the evolution of application models. As a starting point, an informal discussion of the pursued approach to the modelling of evolution of application models is provided. This leads to a first formal definition of an evolving information system, and provides two ways of viewing these systems. One, the evolution of an application model can be modelled by a series of events, modelling the changes from state to state, thus tracking the evolution of the information system. Two, the evolution of an application model can be modelled, as is shown below, by the evolution of the application model elements.

3.2.1 The approach

The three ER schemata, together with the associated action specifications discussed in example 1.3.1 (page 15), correspond to three distinct snapshots of an evolving universe of discourse. Several approaches can be taken to the modelling of this evolution. In this subsection, we discuss some of them.

A first approach is to model the history of application model elements by adding birth-death relations to all object types in the information structure ([SA85]), illustrated in figure 3.2. This approach, however, is very limited as it only enables the modelling of evolution of the population of an information system. For example, the evolution of the object type Recording cannot be modelled in this approach (only its instances).

In this thesis, another approach is taken in that the evolution of an application model is treated as a separate concept, i.e. evolution/time is considered to be part of the meta model. This approach does not exclude the possibility of deriving a view on the evolution of the application model using the first approach, in particular, it is still possible to derive the view given in figure 3.2.

When pursuing the second approach, two major alternatives exist for dealing with the history of application models. The first one is to maintain a version history of application models in their entirety. This alternative leads to a

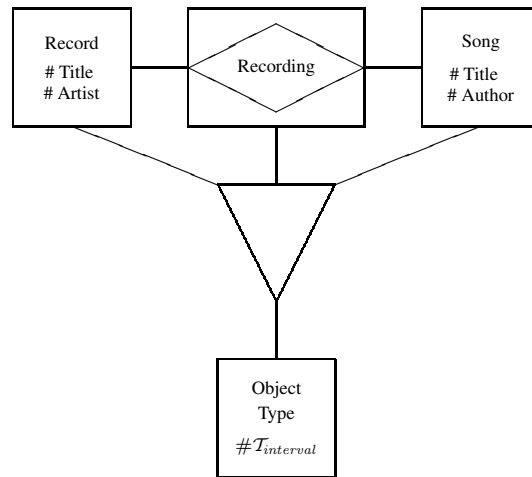


Figure 3.2: Adding history explicitly

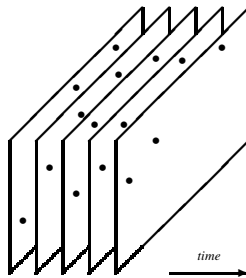


Figure 3.3: Evolution modelled by snapshots

sequence of snapshots of application models, illustrated in figure 3.3. The second alternative, is to keep a version history per element, thus keeping track of the evolution of individual object types, instances, action specifications, etc. This is illustrated in figure 3.4. Each dotted line corresponds to the evolution of one distinct element.

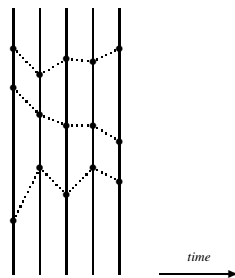


Figure 3.4: Evolution modelled by functions over time

An example of such an element evolution is the evolution of the relation type *Recording* in the rental store from example 1.3.1 (page 15). When CDs are added to its assortment, the version of the application model element *Recording* changes from a relationship type registering songs on *Records*, to a relationship type registering songs on *Media*. The removal of *Records* from the assortment leads to the change of the application model element *Recording* to a relation type registering songs on CDs.

The major advantage of the second alternative is that it enables one to state rules about, and query, the evolution of distinct application model elements. The first alternative clearly does not offer this opportunity, as it does not provide

relations between successive versions of the application model elements. In the theory of evolving application models, we therefore adopt the second alternative.

The snapshot view from the first alternative can be derived by constructing the application model version of any point in time from the current versions of its elements. Consequently, the view on the evolution of populations of the first approach can also be derived. This derivation is illustrated in figure 3.5.

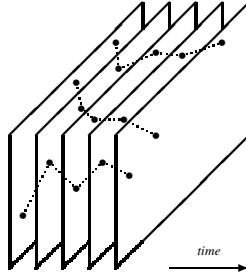


Figure 3.5: Deriving snapshots from element evolutions

The above discussion has a parallel in the description of the history (evolution) of the world. Many approaches are possible. One may choose to describe the evolution of the world as a sequence of snapshots, where each snapshot contains all facts valid at a given point in time. This (highly inefficient) way of describing the history of the world is not used in practice, as one always wants to know the distinct evolution of persons, municipalities, families, laws, countries, etc. Consequently, if one wants to make a complete description of the evolution of the world, the evolutions of all relevant elements of the world (tectonic plates, mountains, persons, ...) have to be described separately.

3.2.2 The formal model

We are now in a position to formally introduce *evolving information systems*. The intention of an evolving information system is to describe an *application model history*. An application model history in its turn, is a set of (*application model*) *element evolutions*. Each element evolution describes the evolution of a specific *application model element*. An element evolution is a partial function assigning to points in time the actual occurrence (version) of that element.

Remark 3.2.1

The difference between valid and event time [SA85], and the ability to correct stored information are not as yet taken into consideration in this chapter; this issue is dealt with in chapter 4. \square

The domain \mathcal{AMH} for application model histories, together with the operations on such histories, are identified by the following components:

1. The set \mathcal{AME} is the domain for the evolvable elements of an application model consisting of all the application model elements (for instance object types, constraints, instantiations, etc.). A formal definition of \mathcal{AME} is provided in section 3.5.
2. Time, essential to evolution, is incorporated into the theory through the algebraic structure $\mathcal{T}_s \triangleq \langle \mathcal{T}, F \rangle$, where \mathcal{T} is a (discrete, totally ordered) *time axis*, and F a set of relations over \mathcal{T} . For the moment, F is assumed to contain the one-step increment function \triangleright , and the comparison relation \leq .

The time axis is the axis along which the application model evolves. With this time axis, an *application model element evolution* can be defined as a (partial) mapping $\mathcal{T} \rightarrow \mathcal{AME}$. The set of *application model histories* is then identified by:

$$\mathcal{AMH} \triangleq \wp(\mathcal{T} \rightarrow \mathcal{AME})$$

In later sections, we pose well-formedness restrictions on such histories.

3. \mathcal{M} is the domain for actions that can be performed of application model histories.
4. The semantics of the actions in \mathcal{M} is provided by the state transition relation on application model histories: $\llbracket \cdot \rrbracket \subseteq \mathcal{M} \times \mathcal{T} \times \mathcal{AMH} \times \mathcal{AMH}$, where $H \llbracket m \rrbracket_t H'$ means: H' may result after applying action m to H at time t . Usually, however, actions are deterministic, rendering a unique H' .

It should be obvious that the semantics of most activity/process modelling techniques is expressible in terms of the $\llbracket \cdot \rrbracket$. The semantics of an action $m \in \mathcal{M}$ is therefore assumed to be provided by the concrete action modelling technique. These semantics can be provided by, for instance, TLA (Temporal Logic of Actions) as defined in [Lam91], or Process Algebra as discussed in [BW90a].

Remark 3.2.2

Other time models are possible, for example, in distributed systems a relative time model might be used. In e.g. [All84], [CR87], [WJL91], and [Cho93] a number of alternative time models can be found. For a general survey on time models, see [RP92]. The linear time model is usually chosen in historical databases (see for example [Sno90]). \square

3.2.3 A dual vision

The performance of an action, on an application model history, at some point in time is referred to as an *event occurrence*. The domain of sequences of such event occurrences is identified by:

Definition 3.2.1 (*event occurrence sequences*)

$$\mathcal{EO} \triangleq \mathcal{T} \rightarrow \mathcal{M}$$

\square

An application model history describes the evolution of an underlying application. A prefix of this history describes the evolution of the application model involved up to some point in time, and forms a *state* of an associated evolving information system. First, we introduce *prefixing* of a single element evolution:

Definition 3.2.2 (*element evolution prefix*)

If $h : \mathcal{T} \rightarrow \mathcal{AME}$ then the prefix of h at time t is:

$$h|_t \triangleq \lambda s. \text{if } s \leq t \text{ then } h(s) \text{ else } h(t) \text{ fi}$$

\square

Some obvious properties, involving idempotence, for history prefixing are:

Proposition 3.2.1 If $h : \mathcal{T} \rightarrow \mathcal{AME}$, then:

$$u \leq t \Rightarrow (h|_t)|_u = h|_u \text{ and } t \leq u \Rightarrow (h|_t)|_u = h|_t$$

The states of an evolving information system, tracking application model history H , are identified by:

Definition 3.2.3 (*evolving information system state*)

If $H \in \mathcal{AMH}$ then the state of H at time t is: $H|_t \triangleq \{h|_t \mid h \in H\}$

\square

Note that each state of an evolving information system is also an application model history ($H|_t \in \mathcal{AMH}$). States are also referred to as *initial histories*. The result of proposition 3.2.1 can be generalised to:

Corollary 3.2.1 If H is an application model history, then:

$$u \leq t \Rightarrow (H|_t)|_u = H|_u \text{ and } t \leq u \Rightarrow (H|_t)|_u = H|_t$$

The evolution of an application model is described by an application model history H . Evolution may also be modelled as a sequence E of event occurrences, specifying subsequent changes to initial histories of the application model, starting from the initial application model. Thus the combination of E and H leads to a dual vision on states of evolving information systems. In the one case, a state results from a set of event occurrences, in the other case, a state is a prefix of an application model history.

A broader view on this duality, relating it to figure 1.9 (page 11), is depicted in figure 3.6. In our way of thinking, a user observes the history of a universe of discourse (H_{uod}), and formulates the observed evolution using a set of event occurrences (E). These event occurrences result, using the Behaves relation, in an application model history (H_{is}) stored in the information system. This latter application model history must be validated (empirically) against the observed history of the universe of discourse.

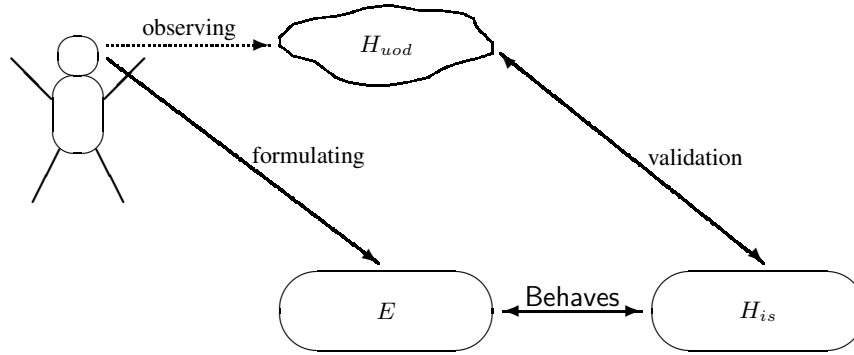


Figure 3.6: A user observing the history of a universe of discourse

Remark 3.2.3

The relation between H and E can be formulated (in a pseudo-formalism) intuitively as:

$$\frac{dH|_t}{dt} = \begin{cases} E(t) & \text{if } E \downarrow t \\ \perp & \text{otherwise} \end{cases}$$

□

The relation between an application model history H , and a set of event occurrences E is captured by the Behaves predicate, stating that every event occurrence must be reflected in the application model history H .

Definition 3.2.4 (well-behaved information system)

Let $E \in \mathcal{EO}$ and $H \in \mathcal{AMH}$, then: $\text{Behaves}(E, H) \triangleq \forall t \in \mathcal{T} \left[H|_t \llbracket E \rrbracket_t^{cwa} H|_{\triangleright t} \right]$, where

$$\begin{aligned} H_1 \llbracket E \rrbracket_t^{cwa} H_2 &\triangleq E \downarrow t \Rightarrow H_1 \llbracket E(t) \rrbracket_t H_2 \wedge \\ &\neg E \downarrow t \Rightarrow H_1 = H_2 \end{aligned}$$

Note that $\llbracket E \rrbracket_t^{cwa}$ makes a closed world assumption on the events that have taken place. If no event occurred ($\neg E \downarrow t$), no action has taken place. One may argue that the notion of ‘frame assumption’ from default logic may be better suited to explain this behaviour. However, we prefer using the term ‘closed world assumption’.

□

Remark 3.2.4

States of an evolving information system are finite approximations of the associated application model history, or semi-formal:

$$\lim_{t \rightarrow \infty} \langle E|_t, H|_t \rangle = \langle E, H \rangle$$

Note that $E|_t$ and $H|_t$ monotonously increase in size, over the course in time. \square

At this point, the states and transitions of an evolving information system are demarcated. Later, well-formedness restrictions are imposed on application model histories, and thus indirectly also on the states of the evolving information system. We will use $\text{IsAMH}(H)$ to denote that H satisfies these restrictions. These restrictions on states imply a restriction on transitions, expressed by the predicate IsEIS :

Definition 3.2.5 (well-behaved proper evolving information system)

Let $E \in \mathcal{EO}$, $H \in \mathcal{AMH}$, then:

$$\text{IsEIS}(E, H) \triangleq \text{Behaves}(E, H) \wedge \forall t \in \mathcal{T} [\text{IsAMH}(H|_t)]$$

\square

Remark 3.2.5

In the course of the life time of an evolving information system, the set of event occurrences will grow. This can be compared to the notion of entropy in thermo dynamics. Entropy is a physical measure for chaos, and cannot decrease in time (see for example [Haw90]). \square

3.3 Generalised Application Models

The application model contains an information structure as well as a number of other elements. The hierarchy of models in figure 1.7 (page 9) describes how an application model is constructed from other (sub)models, however, this hierarchy disregards relations between the submodels. For example, how a population relates to the information structure. These relations are crucial elements of an application model, as they form the fabric of the application model.

The following section provides the formal definition of an application model version, containing all components from the hierarchy of models, and the relationships between them. First, we delimit the state space of the application model versions using an application model universe. An application model version provides a complete description of the state of the information system at some point in time. Such an application model version is bound to the *application model universe* \mathcal{U}_Σ .

Definition 3.3.1

An application model universe \mathcal{U}_Σ is spanned by the tuple:

$$\mathcal{U}_\Sigma \triangleq \langle \mathcal{U}_{\mathcal{IS}}, \mathcal{D}, \Omega, \text{IsPop}, \gamma, \text{IsConstrSet}, \mu, \text{IsActMod}, \llbracket \cdot \rrbracket, \text{IsEvol}, \text{Depends} \rangle$$

\square

The information structure universe $\mathcal{U}_{\mathcal{IS}}$ was introduced in the previous chapter. The other components of the application model universe, and their signatures, are discussed in the remainder of this section.

3.3.1 Domains

The separation between the concrete and the abstract world is provided by the distinction between the information structure \mathcal{I} , and the set of underlying (concrete) domains in \mathcal{D} ([HPW93]). Therefore, an application model version contains a mapping $\text{Dom}_t : \mathcal{L} \rightarrow \mathcal{D}$ providing the relationship between label types and domains in that version. The domain of these assignments is defined as: $\text{Dom} \triangleq \mathcal{L} \times \mathcal{D}$ so $\text{Dom}_t \subseteq \text{Dom}$ such that Dom_t is a functional relation. Some illustrative examples of such domain assignments, in the context of the running rental store example, are: $\text{Times} \mapsto \text{Natno}$ and $\text{Title} \mapsto \text{String}$ where Natno and String are assumed to be (names of) concrete domains.

3.3.2 Instances

The population of an information structure is not, as usual, a partial function that maps object types to sets of values. We consider an instance to be an independent application model element, evolving over the course in time. In our view, an *instance* consists of a value and an associated set of object types. For example, the value 'Brother in Arms' and object types $\{\text{Record}, \text{Medium}\}$ define one instance. The set of instances in an application model version at point in time t is provided by the relation HasTypes_t . The expression $x \text{ HasTypes}_t T$ states that $\langle x, T \rangle$ is an instance, and x is a value with associated types T . The domain for this relation is: $\text{HasTypes} \triangleq \Omega \times \wp^+(\mathcal{O})$ where \wp^+ denotes the power set operation excluding the empty set, and Ω is the set of all allowed values of instances.

Since HasTypes_t is a relation rather than a (partial) function, one value may have associated multiple sets of object types, resulting in different instances with the same value. The reason for this lies in the support of complex generalisation hierarchies. For example, if $\{a_1, a_2\}$ is an instance of both D and E in figure 2.6 (page 30), then this can either be modelled as $\{a_1, a_2\} \text{ HasTypes}_t \{D, E, F, G\}$, or as the instantiations $\{a_1, a_2\} \text{ HasTypes}_t \{D, F\}$ and $\{a_1, a_2\} \text{ HasTypes}_t \{E, G\}$. The difference between these two options comes to the fore, when considering evolution of instances. The second way of modelling allows us to describe the evolution of both instantiations, even when they have the same value ($\{a_1, a_2\}$), separately. A concrete example of such a situation would be when object type A is a set of students, B is the set of students participating in practical groups, and C is the set of students playing soccer. The object types E and D , then correspond to the soccer teams, and the practical groups respectively. Now consider the value $\{a_1, \dots, a_{14}\}$. This value may well be simultaneously a soccer team: $\langle \{a_1, \dots, a_{14}\}, \{E\} \rangle$ and a practical group: $\langle \{a_1, \dots, a_{14}\}, \{D\} \rangle$. Nevertheless, one is still able to model the evolution of both instantiations separately since the soccer team may evolve as a result of a transfer of a player, to: $\langle \{a_1, \dots, a_{13}, b_1\}, \{C\} \rangle$, and the practical group as a result of change in subject by a student to: $\langle \{a_1, \dots, a_{11}, a_{13}, a_{14}\}, \{D\} \rangle$. In such cases, one clearly wants to model the evolution of these instances separately.

The population of an object type in a version, traditionally provided as a function $\mathcal{O} \rightarrow \wp(\Omega)$, can be derived from the association between instances and object types:

$$\text{Pop}_t(x) \triangleq \{v \mid \exists Y [v \text{ HasTypes}_t Y \wedge x \in Y]\}$$

Not all subsets of HasTypes correspond to a proper population. A population of an information structure should adhere to some technique dependent properties. These properties are assumed to be provided (by a concrete modelling technique) by the predicate $\text{IsPop} \subseteq \wp(\mathcal{O}) \times \wp(\text{HasTypes})$. Note that a set of object types O completely determines an information structure. The intuitive thinking behind the expression $\text{IsPop}(O, H)$ is therefore: H is a proper instantiation of the information structure O . The definition of this predicate is given in section 3.5.

3.3.3 Constraints

Most data modelling techniques offer a language for expressing both state and transition oriented constraints. This language describes a set γ of *constraint definitions* over an information structure universe $\mathcal{U}_{\mathcal{TS}}$.

In our theory, we provide the possibility of associating constraint definitions to object types. Constraints are treated as application model elements, that assign constraint definitions to some object types. A constraint c is said to be *owned* by an object type x , if x has assigned a constraint definition by c (so $c \downarrow x$). Ownership of constraint definitions is inherited from one object type to another via the identification hierarchy, however, as in object-oriented data modelling techniques, *overriding* of constraint definitions in identification hierarchies is possible (see for instance [De 91]). Note that due to the inheritance of instances in an identification hierarchy, the only sensible form of overriding is the *strengthening* of constraints, where strengthening means a further reduction of the set of valid populations. The inheritance of constraints is covered formally in the next section.

As an illustration of the assignment of constraints to object types, consider figure 3.7. The data model depicted conforms to NIAM, while the subtype defining rules have been formulated in LISA-D ([HPW93], [HPW94a]). The modelled universe of discourse concerns the administration of planes; as planes have to be replaced in time, the age of a plane is an important attribute. Further, a plane may be registered by an aviation association, in which case it

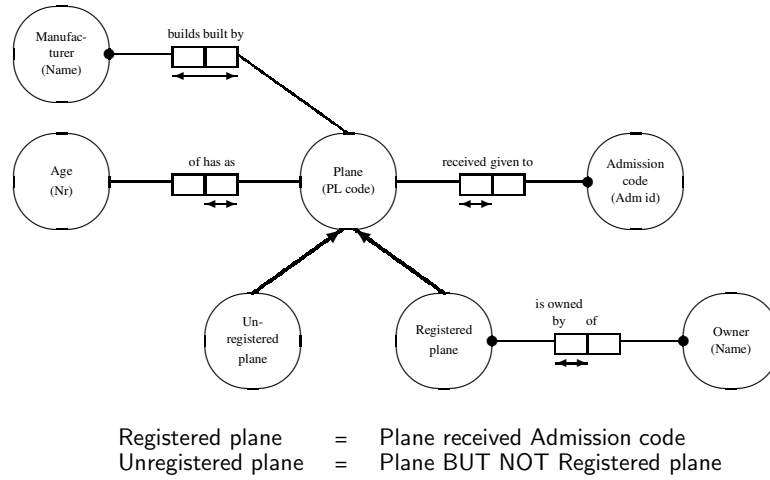


Figure 3.7: Constraint assignment

has an associated admission code. A register of plane ownership is maintained by the administration. The graphical constraints contained in this data model, are assigned to object types in the following way:

- C_1 : Manufacturer \mapsto TOTAL { Manufacturer : builds }
 C_2 : Plane \mapsto UNIQUE { Plane : has as }
 C_3 : Admission code \mapsto TOTAL { Admission code : given to }

All these constraints are owned by a single object type. A more interesting case with respect to inheritance results by adding the following constraint:

All planes must have an associated manufacturer or an age. Unregistered planes must have both.

The object type assignment for this constraint is:

- C_4 : Plane \mapsto TOTAL { Plane : built by, Plane : has as }
 C_4 : Registered plane \mapsto TOTAL { Plane : built by, Plane : has as }
 C_4 : Unregistered plane \mapsto TOTAL { Unregistered plane : built by } AND
 TOTAL { Unregistered plane : has as }

The constraint C_4 is owned by object types Plane, Registered plane and Unregistered plane. Note that constraint definition $C_4(\text{Unregistered plane})$ is more restrictive than $C_4(\text{Plane})$. Finally, as an illustration of a transition oriented constraint, consider the following constraint for planes:

- C_5 : Plane \mapsto (Unregistered plane BEFORE Registered plane) EQUALS Plane
 C_5 : Registered plane \mapsto (Unregistered plane BEFORE Registered plane) EQUALS Plane
 C_5 : Unregistered plane \mapsto (Unregistered plane BEFORE Registered plane) EQUALS Plane

stating that every plane must have been unregistered before being a registered plane.

Remark 3.3.1

Only basic rules governing the assignment of constraints to groups of object types are provided in the general theory. A concrete modelling technique will provide more detailed rules, for instance dealing with problems of multiple inheritance ([CW85]).

For a data modelling technique like PSM, the association of constraints to a group of object types can be used as a further abstraction mechanism for complex schemas ([HPW94a]). The group of object types to which a constraint should/may be associated can be determined by schema objectification provided by PSM, or grouping algorithms clustering object types based on the cardinality of the relationships between these object

types. Examples of such algorithms can be found in [NH89] (the ONF algorithm), [BW92a] and [CH93]. In [CH93] an example is given of how to derive an EER view from a NIAM schema, thus providing a means of hiding details from the NIAM schema. \square

A constraint c , in an application model version, is modelled as a (usually very sparse) partial function $c : \mathcal{O} \rightharpoonup \gamma$. This function provides for every object type a *private* definition of the constraint. Each modelling technique will have its own possibilities to formulate inheritance rules, further governing the mapping of c . The domain for such constraints is identified by: $\mathcal{R} \triangleq \mathcal{O} \rightharpoonup \gamma$. As the enforcement of constraints (in particular transition oriented constraints) can only be done properly in a temporal context, this is postponed to section 3.5. We will regard constraint enforcement as a well-formedness criterion on the evolution of an application model.

Finally, the modelling technique dependent $\text{IsConstrSet} \subseteq \wp(\mathcal{R})$ predicate, is used to designate which sets of constraints are (syntactically and semantically) correct. This predicate can be used for the prevention of inconsistencies in the constraints defined for an information structure.

3.3.4 Methods

The action model part of an application model version is provided as a set of action specifications. The domain for *action specifications* (μ) is determined by the chosen modelling technique for the action model. Analogously to constraint definitions, action specifications are associated to object types. A *method* m is defined as a partial function $m : \mathcal{O} \rightharpoonup \mu$, assigning action specifications to object types. The set of all possible methods is the set of all these mappings: $\mathcal{M} \triangleq \mathcal{O} \rightharpoonup \mu$. The discussion on the, modelling technique dependent, inheritance and grouping mechanism for the ownership of constraints also applies to action specifications.

The modelling technique dependent $\text{IsActMod} \subseteq \wp(\mathcal{M})$ predicate is used to designate what correct action model (being a set of methods) are.

3.3.5 Semantics of action and constraint specifications

The semantics of action specifications and constraint definitions are both defined by the relation $\llbracket \cdot \rrbracket$. Even more, constraint definitions can simply be regarded as a special kind of action specifications, this approach is in line with the approach taken by TLA ([Lam91]). This leads to the following axiom:

$$[\text{AMU1}] \quad \gamma \subseteq \mu$$

A direct result of this axiom is: $\mathcal{R} \subseteq \mathcal{M}$. Next, we focus at the semantics of methods, or rather the underlying action specifications, which are described by $\llbracket \cdot \rrbracket$ as transitions on application model histories.

When performing an action specification on an application model history, certain well-formedness properties of the application model histories, have to be maintained. These well-formedness rules are introduced in subsection 3.5.5, and will lead to the predicate IsAMH . For now, we can only state that the performance of actions should not violate this well-formedness:

$$[\text{AMU2}] \quad H \llbracket m \rrbracket_t H' \wedge \text{IsAMH}(H) \Rightarrow \text{IsAMH}(H')$$

The meaning of an action specification may depend on the history, to date, of an application model. It may, however, not depend on any future behaviour of the application model:

$$[\text{AMU3}] \quad H \llbracket m \rrbracket_t H' \Rightarrow H = H_t$$

Furthermore, the effect of an action specification is completely known after its completion:

$$[\text{AMU4}] \quad H \llbracket m \rrbracket_t H' \Rightarrow H' = H'_{|_{\triangleright t}}$$

It should be noted that action specifications can refer to the complete history H when determining their effects for the new history H' .

From this axiom and corollary 3.2.1 (page 42) immediately follows the following re-formulation of this rule, more closely following the intuition.

Corollary 3.3.1 $H \llbracket m \rrbracket_t H' \Rightarrow \forall_{u>t} [H' = H'_u]$

Combined with axiom AMU2, this leads to:

Corollary 3.3.2 $H \llbracket m \rrbracket_t H' \wedge \text{IsAMH}(H) \Rightarrow \forall_{u>t} [\text{IsAMH}(H'_u)]$.

stating that a well-formed application model history remains a well-formed history at every point in time in the future, after performing an action.

The history of an application model is supposed to be monotonous. Therefore, it is not possible to falsify the history:

[AMU5] $H \llbracket m \rrbracket_t H' \Rightarrow H = H'_t$.

An application model history resulting from a change at some point in time, completely reflects this change. This observation follows from the above axiom in conjunction with AMU4, and corollary 3.2.1 (page 42):

Corollary 3.3.3 $H \llbracket m \rrbracket_t H' \Rightarrow H'_t \llbracket m \rrbracket_t H'_{|>t}$.

Furthermore, when performing an action on a history, the past is entirely copied. This follows by combining axiom AMU3, AMU5, and corollary 3.2.1 (page 42):

Corollary 3.3.4 $H \llbracket m \rrbracket_t H' \Rightarrow \forall_{u \leq t} [H_u = H'_u]$

Constraint definitions are deemed to be a special kind of action specifications, behaving as a guard on application model histories. As a result, constraints are basically predicates. Therefore, the semantics of constraint definitions does not influence the future behaviour of the application model history. This means that two application model histories which have a common past, adhere to the same constraints in that common past:

[AMU6] If $d \in \gamma$ then $H_t \llbracket d \rrbracket_t H_{|>t} \wedge H_t = H'_t \Rightarrow H'_t \llbracket d \rrbracket_t H'_{|>t}$

This axiom implies that $H \llbracket d \rrbracket_t$ (without second argument) is a meaningful expression. Note that the constraint definitions d can still be transition oriented since the entire past is contained in H .

Not all application model histories H are considered to be well-formed evolutions. In section 3.5 some well-formedness rules on the general theory level will be stated. The $\text{IsEvol} \subseteq \mathcal{AMH}$ predicate is employed to cater for modelling technique dependent well-formedness rules on application model evolutions. The IsEvol predicate must be able to determine the correctness of an application model history by looking at a finite prefix of the history:

[AMU7] $\forall_{t \in T} [\text{IsEvol}(H_t)] \Rightarrow \text{IsEvol}(H)$

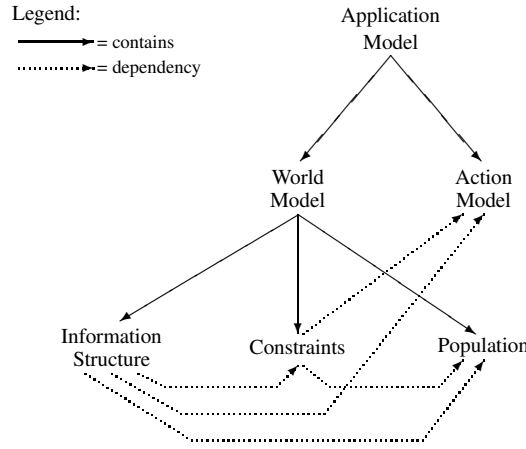


Figure 3.8: Evolution Dependency

3.3.6 Evolution dependency

Normally, some syntactic mechanism in the form of a language or technique is provided for the specialisation of methods (and constraints). For example, for figure 3.7 the specification language LISA-D is used to express non-graphical constraints. The graphical constraints in figure 3.7 form another example of the use of a (graphical!) syntactic mechanism.

Every method and constraint may refer to (use) a number of object types and values from some concrete domain. This relation is provided in the application model universe by the dependency relation *Depends*: $\text{Depends} \subseteq \mu \times (\mathcal{O} \cup \mathcal{D})$. Note that: $\gamma \subseteq \mu$. This relation is modelling technique dependent, but is not subject to evolution. The interpretation of this relation is as follows: x *Depends* y means that if y is not alive in an application model version, then x has no meaning in that version. A consequence is that, in case of evolution of application models, when y evolves to y' , x must be adapted accordingly.

As a concrete example, consider the second action specification from the rental store example (figure 1.14 (page 16)). This action specification depends on object types *Medium*, *Record* and *Frequency*. Further, it depends on domain *Natno* due to the domain assignment: $\text{Frequency} \mapsto \text{Natno}$. If one of the object types, or the domain assignment, is terminated or changed, the action specification must be terminated or changed accordingly.

We now take a closer look at *evolution dependency*, and provide an intuitive background to such dependencies. Most updates (evolution steps) of the application model deal with update of the information base. These ‘normal’ updates of instances of object (entity) types in the information base hardly ever lead to updates of other parts of the application model, unless, for example, the violation of a constraint leads to the conclusion that the violated constraint is wrong. A change of the information structure, however, the addition/deletion/change of an object type for example, almost inevitably implies the update of other parts of the application model. The following *evolution dependencies* can be identified:

1. The constraints in the application model depend on the information structure.

If an object type is deleted, all the constraints defined on that object type, or using that object type in their definition, have also to be deleted or changed. If an object type is added, one probably has to add some appropriate constraints as well.

Furthermore, if a constraint is changed, some action in the action model may have to be adopted as well in order to prevent the action from violating the (changed) constraint.

2. The information base is highly dependent on the information structure as well as the constraints.

If an object type is deleted or changed, all the instances of this object type have also to be deleted or changed. Further, whenever a new constraint is added, or a constraint is replaced by a more limitative one, the population

may have to be updated accordingly, i.e. instances may have become illegal. Therefore, information structure evolution inevitably leads to changes of the information base (population).

3. The action model also depends on the information structure.

Whenever an object type is deleted, all the activities operating on objects of this type also have to be deleted, or changed. If an object type is added, one probably also has to specify appropriate activities operating on the instances of this new object type.

All these dependencies are illustrated in figure 3.8. As stated before, a proper formalisation of the evolution dependency relation, depends on the syntactic and semantic definitions of the modelling techniques used for the application model.

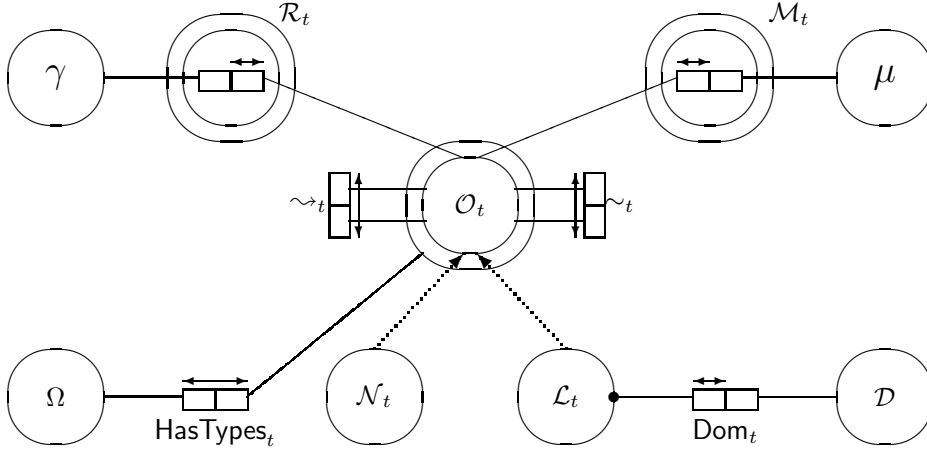


Figure 3.9: A meta model for information structures

3.4 Application Model Versions

The next step of the pyramid depicted in figure 3.1 is introduced in this section, which deals with versions of application models. The description of the evolution of an application, i.e. an application model history, has been introduced as a set of application model element evolutions. Therefore, an application model version can be determined by the actual application model element versions. At this moment we identify the domain for such versions as:

Definition 3.4.1

An application model version over application model universe \mathcal{U}_Σ is defined as:

$$\Sigma_t \triangleq \langle \mathcal{O}_t, \mathcal{R}_t, \mathcal{M}_t, \text{HasTypes}_t, \text{Dom}_t \rangle$$

where $\mathcal{O}_t \subseteq \mathcal{O}$, $\mathcal{R}_t \subseteq \mathcal{R}$, $\mathcal{M}_t \subseteq \mathcal{M}$, $\text{HasTypes}_t \subseteq \text{HasTypes}$ and $\text{Dom}_t \subseteq \text{Dom}$. □

From a version of an application model, the current version $\mathcal{I}_t \triangleq \langle \mathcal{L}_t, \mathcal{N}_t, \sim_t, \rightsquigarrow_t \rangle$ of the information structure can be derived as follows:

$$\begin{aligned} \mathcal{L}_t &\triangleq \mathcal{O}_t \cap \mathcal{L} & \mathcal{N}_t &\triangleq \mathcal{O}_t \cap \mathcal{N} \\ x \sim_t y &\triangleq x \sim y \wedge x, y \in \mathcal{O}_t & x \rightsquigarrow_t y &\triangleq x \rightsquigarrow y \wedge x, y \in \mathcal{O}_t \end{aligned}$$

A meta model giving an overview of the components of an application model version, is given in figure 3.9. A population in this meta model corresponds to one application model version. The model conforms to the PSM data modelling technique.

Every application model version must adhere to certain rules of well-formedness. Some of these rules are modelling technique dependent, and therefore outside the scope of this chapter. Nevertheless, some general rules about application model versions can be stated.

3.4.1 Active and living object types

An object type x is called *alive* at a certain point in time t , if it is part of the application model version at that point in time ($x \in \mathcal{O}_t$). Further, an object type x is termed *active* at a certain point in time, if there is an *instance typing* X at t such that $x \in X$. We call X an instance typing at t if $\exists_v [v \text{ HasTypes}_t X]$.

A first rule of well-formedness states that every active object type must also be alive. This rule can be popularised as: ‘I am active, therefore I am alive’, or: ‘ghost object types do not exist’. It is formalised as:

[AMV1] (*active life*) If X is an instance typing at t , then: $X \subseteq \mathcal{O}_t$

The next rule of well-formedness states that sharing an instance at any point in time, is to be interpreted as a proof of type relatedness:

[AMV2] (*active relatedness*) If X is an instance typing at some point in time, then: $x, y \in X \Rightarrow x \sim y$

From the very nature of the root relation it follows that instances are inherited in the direction of the roots. For instance, in figure 1.14 (page 16) Medium is the root of the Medium, Record, CD subtyping hierarchy, and every instance of Record or CD is also an instance of Medium. As a result, every instance of an object type is also an instance of its ancestors (if any):

[AMV3] (*foundation of activity*) If X is an instance typing at some point in time, then the property A_X , defined by $A_X(x) \triangleq x \in X$, is reflected by \leadsto .

Applying the foundation schema (corollary 2.3.1 (page 28)) to this axiom shows the presence of roots in instance typings:

Corollary 3.4.1 (*active roots*) If X is an instance typing, then A_X is reflected by RootOf.

In most traditional data modelling techniques (such as ER, NIAM and FORM) each type hierarchy has an unique root. As a consequence, each instance typing contains an unique root. Some data modelling techniques (such as PSM), however, allow type hierarchies with multiple roots (see figure 2.6 (page 30)). For such modelling techniques, the following axiom guarantees an unique root for each instance typing.

[AMV4] (*unique root*) If X is an instance typing at some point in time, and $x, y \in X$ then: $\text{IsRoot}(x) \wedge \text{IsRoot}(y) \Rightarrow x = y$

This axiom leads to the following strengthening of corollary 3.4.1 (page 50):

Lemma 3.4.1 (*active root*) If X is an instance typing at some point in time, then:

$$y \in X \Rightarrow \exists!_x [x \in X \wedge x \text{ RootOf } y]$$

Axiom AMV3 also has a structural pendant: every living object type is accompanied by one of its ancestors (if any). This is stipulated in the following axiom:

[AMV5] (*foundation of live*) The property L_t identified by $L_t(x) \triangleq x \in \mathcal{O}_t$, is reflected by \leadsto at all points in time t .

Note that axiom AMV5 cannot be derived from axiom AMV3. The reason is that a non-root object type may be alive, yet have no associated instance. By applying the foundation schema (corollary 2.3.1 (page 28)) on axiom AMV5 we get:

Corollary 3.4.2 (*living roots*) For all points in time t , property L_t is reflected by RootOf.

3.4.2 Well-formed concretisation

In a valid application model version each label type is *concretised* by associating a domain. Therefore, the domain providing function Dom_t is a (total) function from alive label types to domains:

[AMV6] (*full concretisation*) $\text{Dom}_t : \mathcal{L}_t \rightarrow \mathcal{D}$

The domain assignment respects the inheritance hierarchy:

[AMV7] (*domain inclusion*) If $l_1, l_2 \in \mathcal{L}$, then: $l_1 \rightsquigarrow l_2 \Rightarrow \text{Dom}_t(l_2) \subseteq \text{Dom}_t(l_1)$

Further, the instances of label types must adhere to this domain assignment:

[AMV8] (*strong typing*) If $v \text{ HasTypes}_t X$ and $v \in \bigcup \mathcal{D}$ then: $x \in X \Rightarrow v \in \text{Dom}_t(x)$

3.4.3 Constraints and methods

Methods, and consequently constraints, are defined as mappings from object types to method and constraint definitions respectively. This implies that object types, owning a constraint or a method, must be alive.

[AMV9] (*alive definitions*) If $w \in \mathcal{R}_t \cup \mathcal{M}_t$ then: $\text{dom}(w) \subseteq \mathcal{O}_t$

Due to inheritance of properties, if a constraint is defined for an ancestor object type, it is also defined for its offspring.

[AMV10] (*inheritance of definitions*) For all $w \in \mathcal{R}_t \cup \mathcal{M}_t$, the property D_w defined by $D_w(x) \triangleq w \downarrow x$, is preserved by \rightsquigarrow .

Note that the inheritance direction for populations, is reverse to the inheritance direction for methods (and constraints). Since every instance from a non-root object type is inherited downwards in the identification hierarchy towards the root object types, constraints on child-object types should be at least as restrictive:

[AMV11] (*strengthening of constraints*)

If $c \in \mathcal{R}_t$, then: $x \rightsquigarrow y \wedge c \downarrow x, y \Rightarrow c(y) \Vdash_{\mathcal{U}_\Sigma} c(x)$

where $d_1 \Vdash_{\mathcal{U}_\Sigma} d_2$ is defined as: $d_1 \Vdash_{\mathcal{U}_\Sigma} d_2 \triangleq \forall_{t,H} [H \llbracket d_1 \rrbracket_t \Rightarrow H \llbracket d_2 \rrbracket_t]$

The intuitive meaning of $d_1 \Vdash_{\mathcal{U}_\Sigma} d_2$ is: d_1 is at least as restrictive as d_2 (see also [Hof93]).

The motivation for the following axiom lies in the following observation. The definition of a constraint or a method refers to a set of object types concrete domains. Therefore, if a method or constraint definition is alive, then all these referred items should be alive at that same moment.

[AMV12] (*dangling references*)

If $w \in \mathcal{R}_t \cup \mathcal{M}_t$ then: $w(x) \text{ Depends } y \Rightarrow y \in \mathcal{O}_t \cup \mathcal{D}_t$ where $\mathcal{D}_t = \text{ran}(\text{Dom}_t)$.

Further, a method or constraint at least refers to the object type for which it is defined:

[AMV13] (*self referencing*) If $w \in \mathcal{R}_t \cup \mathcal{M}_t$ then: $w \downarrow x \Rightarrow w(x) \text{ Depends } x$

3.4.4 Populations of information structures

A special part of an application model version is its population. As stated before, this population can be derived from the relation HasTypes_t . For reasons of completeness, we now provide this definition more formally:

Definition 3.4.2

The population of an information structure at any point in time, is defined by:

$$\text{Pop} : \mathcal{T} \rightarrow (\mathcal{O} \rightarrow \wp(\Omega)) \text{ with } \text{Pop}_t(x) \triangleq \{v \mid \exists_Y [v \text{ HasTypes}_t Y \wedge x \in Y]\}$$

□

It turns out to be convenient (see chapter 8) to have an overview of all instances that ever lived. We refer to this population as the *extra-temporal population*.

Definition 3.4.3

The extra-temporal population of an application model is identified by:

$$\text{Pop}_\infty : \mathcal{O} \rightarrow \wp(\Omega) \text{ with } \text{Pop}_\infty(x) \triangleq \bigcup_{t \in \mathcal{T}} \text{Pop}_t(x)$$

□

Axiom AMV3 relates instances to the object type hierarchy. This leads to the following property for populations:

Lemma 3.4.2 (*population distribution*) Every instance of an object type, is also instance of one of its roots:

$$\text{Pop}_t(x) \subseteq \bigcup_{y: y \text{ RootOf } x} \text{Pop}_t(y)$$

Proof:

Let $i \in \text{Pop}_t(x)$ then:

$$\begin{aligned} i \in \text{Pop}_t(x) & \equiv \{\text{definition of Pop}_t\} \\ \exists_X [i \text{ HasTypes}_t X \wedge x \in X] & \Rightarrow \{\text{corollary 3.4.1 (page 50)}\} \\ \exists_{y, X} [i \text{ HasTypes}_t X \wedge x, y \in X \wedge y \text{ RootOf } x] & \equiv \{\text{definition of Pop}_t\} \\ \exists_y [y \text{ RootOf } x \wedge i \in \text{Pop}_t(y)] & \equiv \{\text{elaborate}\} \\ i \in \bigcup_{y: y \text{ RootOf } x} \text{Pop}_t(y) & \end{aligned}$$

□

The result of the previous lemma can be generalised to extra-temporal populations:

Corollary 3.4.3

$$\text{Pop}_\infty(x) \subseteq \bigcup_{y: y \text{ RootOf } x} \text{Pop}_\infty(y)$$

Proof:

Let $v \in \text{Pop}_\infty(x)$ then:

$$\begin{aligned}
 v \in \text{Pop}_\infty(x) &\equiv \{\text{definition of } \text{Pop}_\infty\} \\
 \exists_t [v \in \text{Pop}_t(x)] &\Rightarrow \{\text{lemma 3.4.2 (page 52)}\} \\
 \exists_t \left[v \in \bigcup_{y:y \text{ RootOf } x} \text{Pop}_t(y) \right] &\equiv \{\text{definition of } \text{Pop}_\infty\} \\
 v \in \bigcup_{y:y \text{ RootOf } x} \text{Pop}_\infty(y) &
 \end{aligned}$$

□

Next we focus on strong typing, which is considered to be a property to hold on each moment: if $x \not\sim y$, then their populations may never share instances. The following axiom is sufficient to guarantee this property, as we will show in theorem 3.4.2.

[AMV14] (*exclusive root population*) If $\text{IsRoot}(x)$ and $\text{IsRoot}(y)$ then:

$$x \not\sim y \Rightarrow \text{Pop}_\infty(x) \cap \text{Pop}_\infty(y) = \emptyset$$

If roots are not type related, then their extra-temporal populations are disjoint.

Using the following theorem the nature of type relatedness, captured for roots in the above axiom, is generalised to object types in general:

Theorem 3.4.1 (*exclusive population*) If $x \not\sim y$ then

$$\bigcup_{z:z \text{ RootOf } x} \text{Pop}_\infty(z) \cap \bigcup_{z:z \text{ RootOf } y} \text{Pop}_\infty(z) = \emptyset$$

The populations of object types which are not type related, have no values in common.

Proof:

$$\begin{aligned}
 x \not\sim y &\equiv \{\text{theorem 2.3.4 (page 30)}\} \\
 \neg \exists_{z_1, z_2} [z_1 \text{ RootOf } x \wedge z_2 \text{ RootOf } y \wedge z_1 \sim z_2] &\Rightarrow \{\text{elaborate}\} \\
 \forall_{z_1, z_2} [z_1 \text{ RootOf } x \wedge z_2 \text{ RootOf } y \Rightarrow z_1 \not\sim z_2] &\Rightarrow \{\text{axiom AMV14 (page 53)}\} \\
 \forall_{z_1:z_1 \text{ RootOf } x, z_2:z_2 \text{ RootOf } y} [\text{Pop}_\infty(z_1) \cap \text{Pop}_\infty(z_2) = \emptyset] &\Rightarrow \{\text{elaborate}\} \\
 \bigcup_{z:z \text{ RootOf } x} \text{Pop}_\infty(z) \cap \bigcup_{z:z \text{ RootOf } y} \text{Pop}_\infty(z) = \emptyset &
 \end{aligned}$$

□

The main typing theorem is derived from lemma 3.4.2 (page 52) and theorem 3.4.1 (page 53).

Theorem 3.4.2 (*strong typing theorem*) $x \not\sim y \Rightarrow \text{Pop}_\infty(x) \cap \text{Pop}_\infty(y) = \emptyset$

We are now in a position to define what constitutes a well-formed application model version. Let $\Sigma_t = \langle \mathcal{O}_t, \mathcal{R}_t, \mathcal{M}_t, \text{HasTypes}_t, \text{Dom}_t \rangle$, then:

$$\begin{aligned}
 \text{IsAM}(\Sigma_t) &\triangleq \text{IsWorldModel}(\mathcal{O}_t, \mathcal{R}_t, \text{HasTypes}_t) \wedge \text{IsActMod}(\mathcal{M}_t) \wedge \\
 &\quad \Sigma_t \text{ adheres to the AMV axioms}
 \end{aligned}$$

where $\text{IsWorldModel}(\mathcal{O}_t, \mathcal{R}_t, \text{HasTypes}_t) \triangleq \text{IsSch}(\mathcal{O}_t) \wedge \text{IsPop}(\mathcal{O}_t, \text{HasTypes}_t) \wedge \text{IsConstrSet}(\mathcal{R}_t)$.

In the next section, this predicate is used to define what constitutes a proper application model history (IsAMH).

3.5 Evolution of Application Models

The evolution of an application model is described by the evolution of its elements. The set \mathcal{AME} was introduced as the set of all evolvable elements of an application model. Its formal definition in terms of components of \mathcal{U}_Σ is:

Definition 3.5.1

Application model elements: $\mathcal{AME} \triangleq \mathcal{O} \cup \mathcal{R} \cup \mathcal{M} \cup \text{HasTypes} \cup \text{Dom}$ □

An application model element evolution is defined as a partial function, assigning actual version of application model elements to points in time. Note that the type relatedness and root relationships are defined for the evolution state space as a whole, and are therefore, not subject to any evolution. In this section we present a set of well-formedness rules for application model histories. In the remainder of this section, H is some (fixed) application model history.

3.5.1 Separation of element evolution

The first rule of well-formedness states that the evolution of application model elements is bound to element classes. For example, an object type may not evolve into a method, and a constraint may not evolve into an instance. This is formalised in the following axiom:

[EW1] (*evolution separation*) If $X \in \{\mathcal{O}, \mathcal{R}, \mathcal{M}, \text{HasTypes}, \text{Dom}\}$, and $h \in H$ then:

$$h(t) \in X \Rightarrow \text{ran}(h) \subseteq X$$

As a result, an application model history can be partitioned into the history of its object types, its constraints, its methods, its populations, and its concretisations (of label types):

Definition 3.5.2

$$\begin{aligned} \text{object type histories: } H_{type} &\triangleq \{h \in H \mid \exists_t [h(t) \in \mathcal{O}]\} \\ \text{constraint histories: } H_{constr} &\triangleq \{c \in H \mid \exists_t [c(t) \in \mathcal{R}]\} \\ \text{method histories: } H_{meth} &\triangleq \{a \in H \mid \exists_t [a(t) \in \mathcal{M}]\} \\ \text{population histories: } H_{pop} &\triangleq \{g \in H \mid \exists_t [g(t) \in \text{HasTypes}]\} \\ \text{concretisation histories: } H_{dom} &\triangleq \{d \in H \mid \exists_t [d(t) \in \text{Dom}]\} \end{aligned}$$

□

3.5.2 Deriving application model versions

An application model version $\Sigma_t(H) \triangleq \langle \mathcal{O}_t, \mathcal{R}_t, \mathcal{M}_t, \text{HasTypes}_t, \text{Dom}_t \rangle$ at a given point in time t is easily derived from an application model history H . This derivation is defined by deriving the five main components, which determine an application version, from H :

Definition 3.5.3

$$\begin{aligned} \text{object types: } \mathcal{O}_t &\triangleq \{h(t) \mid h \in H_{type} \wedge h \downarrow t\} \\ \text{constraints: } \mathcal{R}_t &\triangleq \{c(t) \mid c \in H_{constr} \wedge c \downarrow t\} \\ \text{methods: } \mathcal{M}_t &\triangleq \{a(t) \mid a \in H_{meth} \wedge a \downarrow t\} \\ \text{population: } \text{HasTypes}_t &\triangleq \{g(t) \mid g \in H_{pop} \wedge g \downarrow t\} \\ \text{concretisations: } \text{Dom}_t &\triangleq \{d(t) \mid d \in H_{dom} \wedge d \downarrow t\} \end{aligned}$$

□

Figure 3.10 provides a (meta) data model, that relates all concepts defined sofar. The data model depicted, again, conforms to the PSM modelling technique. Readers who are unfamiliar with the used graphical notations are advised to refer to section 5.2.

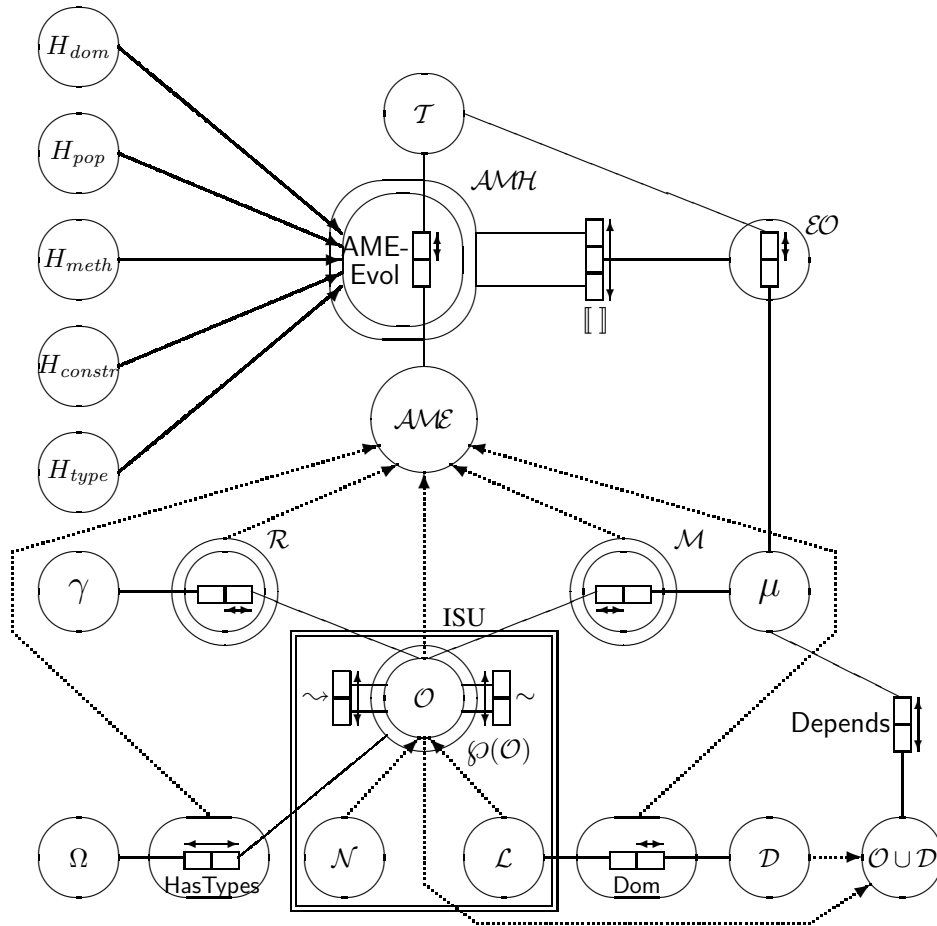


Figure 3.10: A meta model for the evolution theory

3.5.3 Enforcing constraints

As a next well-formedness rule for evolution of application models, we consider constraint enforcement. Every constraint in an application model version must hold, however, constraints do not cause restrictions beyond their lifetime. The intuition behind the limitation of the scope of a constraint, is the *closed world assumption* applied to an application model history, with respect to changes outside the lifetime of the constraint.

[EW2] (*constraints hold*) For all $c \in H_{\text{constr}}$: $c \downarrow t \Rightarrow H_{\text{birth}(c,t)|t} \llbracket c(t) \rrbracket_t$ where:

$$\begin{aligned} \text{birth}(c, t) &\triangleq \min \{b \mid \forall_{b \leq u \leq t} [c(u) = c(t)]\} \\ H_{b|t} &\triangleq \{h_{b|t} \mid h \in H\} \\ h_{b|t} &\triangleq \lambda u. \text{if } u < b \text{ then } h(b) \text{ else } h_t(u) \text{ fi} \end{aligned}$$

Remark 3.5.1

In programming, it is considered bad practice to write self modifying code, i.e. a program that modifies itself.

Analogously, one could formulate a rule stating that actions in an application model should only have effect on the population. Any structural changes (actions, information structure, ...) should thus be performed by an action explicitly specified by the user (an update request). This can be formulated as:

$$\text{If } m \in \mathcal{M}_t \text{ then } H_t \llbracket m \rrbracket_t H_{\triangleright t} \Rightarrow (H - H^{\text{Pop}})_t = (H - H^{\text{Pop}})_{\triangleright t}.$$

The H^{Pop} is the restriction of H to the evolution of the elements of HasTypes .

□

3.5.4 An example of element evolution

The following table gives examples of four object type evolutions ($h_1, h_2, h_3, h_4 \in H_{\text{type}}$) from the rental store (see subsection 1.3.2 (page 14)). In this example, h_1 describes the evolution of the concept of recording, being initially a recording of a song on a record, later changing to the recording of a song on a medium, and finally being a recording of a song on a CD with some associated quality. Similarly, h_2, h_3 and h_4 describe the evolution of Records, Media and CDs.

H_{type}	h_1	h_2	h_3	h_4
t_1	o_4	o_1	—	—
t_2	o_4	o_1	—	—
t_3	o_5	o_1	o_2	o_3
t_4	o_5	o_1	o_2	o_3
t_5	o_6	—	—	o_3

where $t_1, \dots, t_5 \in \mathcal{T}$, and $o_1, \dots, o_6 \in \mathcal{O}$ are object types, such that:

o_1	=	'Entity type: Record'	o_4	=	'Fact type: Recording of Song on Record'
o_2	=	'Entity type: Medium'	o_5	=	'Fact type: Recording of Song on Medium'
o_3	=	'Entity type: CD'	o_6	=	'Fact type: Recording of Song on CD with Quality'

Note that the evolution step (from figure 1.13 to figure 1.14 (page 16)) takes place at point in time t_4 . Two example instance evolutions, obeying the above schema evolution, are $g_1, g_2 \in H_{\text{pop}}$. The instance evolution g_1 describes the evolution of the Record 'Brothers in Arms', whereas g_2 describes the evolution of the recording of the song 'Brothers in Arms' on the Record with the same name.

H_{pop}	g_1	g_2
t_1	$\langle i_1, \{o_1\} \rangle$	—
t_2	$\langle i_1, \{o_1\} \rangle$	$\langle i_2, \{o_4\} \rangle$
t_3	$\langle i_1, \{o_1, o_2\} \rangle$	$\langle i_3, \{o_5\} \rangle$
t_4	$\langle i_1, \{o_1, o_2\} \rangle$	$\langle i_3, \{o_5\} \rangle$
t_5	—	$\langle i_4, \{o_6\} \rangle$

where $i_1, \dots, i_5 \in \Omega$ are instances such that:

i_1	=	'Brothers in Arms'
i_2	=	$\langle \text{Song: 'Brothers in Arms', Record: 'Brothers in Arms'} \rangle$
i_3	=	$\langle \text{Song: 'Brothers in Arms', Medium: 'Brothers in Arms'} \rangle$
i_4	=	$\langle \text{Song: 'Brothers in Arms', CD: 'Brothers in Arms', Quality: 'AAD'} \rangle$

The interpretation of this table leads to:

1. $g_1(t_4) = \langle i_1, \{o_1, o_2\} \rangle$ means: 'Brothers in Arms' is both a Record and a Medium at t_4
2. $g_2(t_2) = \langle i_2, \{o_4\} \rangle$ means: Song 'Brothers in Arms' is Recorded on Record 'Brothers in Arms' at t_2
3. $g_2(t_3) = \langle i_3, \{o_5\} \rangle$ means: Song 'Brothers in Arms' is Recorded on Medium 'Brothers in Arms' at t_3

As an illustration of how queries about these data can be formulated, suppose we are interested in all object types having instances at time t_2 . This can be formulated by the following Elisa-D (which is introduced in chapter 7 as a way of communicating with an evolving information system) query:

LIST Object type having instances AT Time t_2
 describing a path through the disclosure structure, which integrates the application level and meta level. The following example asks for all Records, ever stocked by the Rental Store, containing the song 'Brothers in Arms':

LIST Record has Recording of Song 'Brothers in Arms'

The keywords has and of in the above expression connect object types to relation types.

3.5.5 Evolution of the identification hierarchy

So far we have discussed the well-formedness of the evolution of application model elements, however, as a result of object type evolution, the identification hierarchy will also evolve. This evolution is not completely free, some conservatism with respect to such evolution is appropriate. The following rules should be interpreted as *plausibility rules*, which may be ignored in some situations. Firstly, the order in the identification hierarchy should not change in one step:

[EW3] (monotonous ancestors) If $h_1, h_2 \in H_{type}$ and $h_1, h_2 \downarrow t, \triangleright t$ then:

$$h_1(t) \rightsquigarrow h_2(t) \wedge h_1(\triangleright t) \sim h_2(\triangleright t) \Rightarrow h_1(\triangleright t) \rightsquigarrow h_2(\triangleright t)$$

This axiom is based on 'taste', and one may argue that a weaker formulation such as:

$$h_1(t) \text{ RootOf } h_2(t) \wedge h_1(\triangleright t) \sim h_2(\triangleright t) \Rightarrow h_1(\triangleright t) \text{ RootOf } h_2(\triangleright t)$$

would be more appropriate. A direct consequence of this axiom is that all ancestors of an object type have to be terminated when this object type is promoted to be a root object type:

Lemma 3.5.1 If $h_1, h_2 \in H_{type}$, $h_1 \downarrow t$ and $h_2, h_2 \downarrow \triangleright t$ then

$$h_1(t) \rightsquigarrow h_2(t) \wedge h_1(\triangleright t) \sim h_2(\triangleright t) \wedge \text{IsRoot}(h_2(\triangleright t)) \Rightarrow \neg h_1 \downarrow \triangleright t$$

Proof:

Let $h_1(t) \rightsquigarrow h_2(t) \wedge h_1(\triangleright t) \sim h_2(\triangleright t) \wedge \text{IsRoot}(h_2(\triangleright t))$ and $h_1 \downarrow \triangleright t$, then:

$$h_1(t) \rightsquigarrow h_2(t) \wedge h_1(\triangleright t) \sim h_2(\triangleright t) \wedge \text{IsRoot}(h_2(\triangleright t)) \wedge h_1 \downarrow \triangleright t \quad \Rightarrow \{ \text{axiom EW3} \}$$

$$h_1(\triangleright t) \rightsquigarrow h_2(\triangleright t) \wedge \text{IsRoot}(h_2(\triangleright t)) \quad \Rightarrow \{ \text{definition of IsRoot} \}$$

falsum

□

The following rule for identification hierarchy evolution states that the type-instance relation (derived from the relation HasTypes) is to be maintained in the course of evolution. This leads to the axiom of guided evolution:

[EW4] (guided evolution) If $g \in H_{pop}$ and $g \downarrow t, \triangleright t$ then

$$\exists_{h \in H_{type}} [h(t) \sim \text{Types}_t(g) \wedge h(\triangleright t) \sim \text{Types}_{\triangleright t}(g)]$$

where $x \sim Y$ is defined as $\exists_{y \in Y} [x \sim y]$.

The types associated with an instance evolution g , at point in time t , are introduced by:

$$\text{Types}_t(g) \triangleq \{ X \mid g(t) \text{ HasTypes}_t X \}$$

We can now finally, as promised, define formally:

Definition 3.5.4

$$\text{IsAMH}(H) \triangleq \forall_{t \in \mathcal{T}} [\text{IsAM}(\Sigma_t)] \wedge \text{IsEvol}(H) \wedge H \text{ adheres to the EW axioms}$$

□

A direct result of this definition is the following lemma.

Lemma 3.5.2 $\forall_{t \in \mathcal{T}} [\text{IsAMH}(H|_t)] \Rightarrow \text{IsAMH}(H)$

Proof:

Let $\forall_{t \in \mathcal{T}} [\text{IsAMH}(H|_t)]$, then the definition of IsAMH allows us to split this in three parts:

- $\forall_{t \in \mathcal{T}} [\text{IsAM}(\Sigma_t)]$ follows directly from proposition 3.2.1 and $\forall_{t \in \mathcal{T}} [\text{IsAMH}(H|_t)]$,
- $\text{IsEvol}(H)$ follows from axiom AMU7,
- “ H adheres to the EW axioms” follows from proposition 3.2.1, the fact that the EW axioms only refer to application model versions, and: $\forall_{t \in \mathcal{T}} [\text{IsAMH}(H|_t)]$.

□

Note that not one can not prove for any property P : $\forall_{t \in \mathcal{T}} [P(H|_t)] \Rightarrow P(H)$.

For instance: $P(H) = “H$ only contains finitely many changes to the application model” can not be proven.

3.6 Conclusions

In this chapter, we provided a general theory for the evolution of application models. This theory tries to abstract from details of underlying modelling techniques.

We introduced two classes of new axioms, the AMV axioms defining what a correct application model version (IsAM) is, and the EW axioms defining what a correct application model history (IsAMH) is. The pyramid depicted in figure 3.1 (page 38) provides an overview of the axiom classes defined in this chapter, and the previous chapter, and their interrelationships.

In chapter 5 and chapter 7, the general theory here presented is applied to the Hydra data modelling framework consisting of PSM, LISA-D and Task Structures.

Chapter 4

A Taxonomy of Update in Information Systems

The figure, the car and its door handle were all on a planet called Earth, a world whose entire entry in the Hitch Hiker's Guide to the Galaxy comprised the two words 'Mostly harmless'.

*From: "So Long and Thanks for all the Fish",
Douglas Adams, Pan Books Ltd.*

4.1 States of Application Models

Information systems are meant to fulfill the information needs of organisations. As both information needs and stored information evolve in the course of time, information systems have to be updated from time to time. In traditional information systems the processing of updates is a non-trivial aspect. In the field of evolving information systems this is even more the case ([FOP92b]). The aim of this chapter is therefore, the introduction of a framework for *update* in evolving information systems. In the previous chapter, we ignored the difference between *valid time* and *transaction time* ([Sno90]), these different time axes are taken into proper consideration, in this chapter

As argued in [FOP92c], update in traditional information systems can be regarded as a degeneration of update in evolving information systems. This is due to the fact that the requirements of traditional information systems, with respect to update, are less demanding than those of evolving information systems (see chapter 1). As a result, the framework presented is also applicable to traditional information systems.

A formal model is provided, prior to the introduction of the framework, by which our way of thinking is exposed regarding the relation between states of the universe of discourse, and the information system. One of the major requirements (see subsection 1.3.1) for evolving information systems is that no information may be lost. This implies that no application model element may be deleted. Instead, its behaviour in time must be maintained. In section 3.2 we stated that the history of application model elements is kept by modelling its behaviour over the course of time as a function. These functions provide the valid versions of elements at distinct points in time, and are referred to as application model element evolutions. A set of these element evolutions specifies an application model history H such that $H \in \mathcal{AMH}$.

In general, when modelling a universe of discourse, it is assumed that a set of *states* can be recognised in the universe of discourse, and that there are a number of action specifications that result in a change of state referred to as *state transitions* ([HPW92a], [FOP92a]). This approach is referred to as the state-transition model. Further, we assume that a universe of discourse has a unique *starting state*. In mathematical terms, a universe of discourse $U \circ D$ consists

of a set \mathcal{S}_{uod} of states, a binary relation \mathcal{T}_{uod} over states, and an initial state $s_{uod}^0 \in \mathcal{S}_{uod}$, leading to:

$$U\circ D \triangleq \langle \mathcal{S}_{uod}, \mathcal{T}_{uod}, s_{uod}^0 \rangle$$

The goal of modelling $U\circ D$ is the construction of a formal description H . This H is the application model history of $U\circ D$ in terms of some underlying formalism. A component \mathcal{S}_{am} specifying \mathcal{S}_{uod} , a component \mathcal{T}_{am} specifying \mathcal{T}_{uod} , and a state s_{am}^0 corresponding to the initial state s_{uod}^0 , can be derived from this specification. Using the way application model histories are modelled in chapter 3, this leads to:

$$\begin{aligned} \mathcal{S}_{am} &\triangleq \{ \Sigma_t \mid t \in \mathcal{T} \} \\ s_{am}^0 &\triangleq \Sigma_{\min(\mathcal{T})} \\ \mathcal{T}_{am} &\triangleq \{ \langle \Sigma_t, \Sigma_{\triangleright t} \rangle \mid \Sigma_t \neq \Sigma_{\triangleright t} \wedge t \in \mathcal{T} \} \end{aligned}$$

where Σ_t is derived from an application model history H .

Since the application model history should be a correct specification of the evolution of the universe of discourse, every *event* in the universe of discourse should be reported to the information system by an *update request* (see figure 1.11 (page 14)). In traditional information systems an update request is usually considered to be either an addition, deletion or modification of (pieces of) information contained in the application model. For evolving information systems, as well as for temporal or historical information systems ([SA86]), this traditional notion of update must be revised. The kinds of updates needed for evolving information systems (and historical information systems) are discussed in the following sections.

Every update request leads to a set of *birth*, *death* and *change transitions* of elements in the application model state. A birth transition introduces a new application model element, a death transition removes an application model element from the state, and a change transition replaces one element with another. In *snapshot information systems*, a birth transition of an application model element can be realised by an addition, and a death transition by a deletion. In both cases, the ‘old’ application model states are lost. As a result, a *change* cannot be modelled in a snapshot information system.

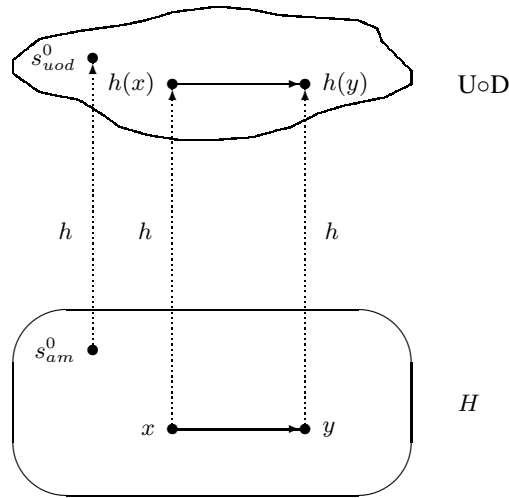


Figure 4.1: A correct specification

The main requirement for specification H is that it behaves like $U\circ D$. This requirement can be visualised by a partial function h , relating the states from \mathcal{S}_{am} to the (real) states \mathcal{S}_{uod} from $U\circ D$, such that h shows this similarity. This situation is depicted in figure 4.1. Such a function is called a partial homomorphism ([HPW92a]).

Definition 4.1.1

We call h a partial homomorphism between H and $U\circ D$ if

1. h is a (partial) function $h : \mathcal{S}_{am} \rightarrow \mathcal{S}_{uod}$

2. transitions commute under h :

$$\forall s, t \in \mathcal{S}_{am} \left[\langle s, t \rangle \in \mathcal{T}_{am} \iff \langle h(s), h(t) \rangle \in \mathcal{T}_{uod} \right]$$

3. h maps the initial state of the specification onto the initial state of $U \circ D$:

$$h(s_{am}^0) = s_{uod}^0$$

If h is surjective, h is called an epimorphism between H and $U \circ D$. □

If each state of $U \circ D$ is captured by the function h , we call H a *correct specification* with respect to $U \circ D$. In this case, the function h should be surjective, and is called an epimorphism. In practice, the function h is provided by the update requests of the users of the information system. As you would expect from the evolving information system paradigm (subsection 1.2.2), users observe events in the universe of discourse and report them to the information system, and by reporting these events the homomorphism between H and $U \circ D$ is maintained. This situation is illustrated in figure 4.2.

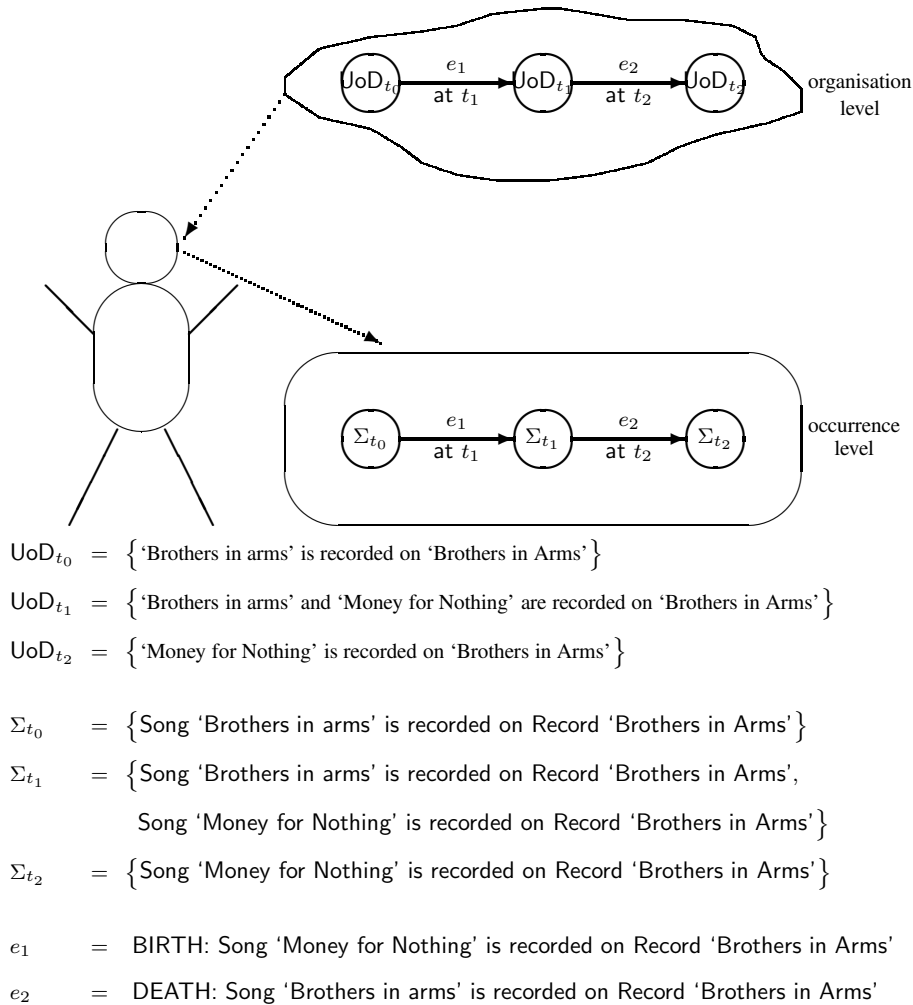


Figure 4.2: Users observe and report events occurring in the universe of discourse

A taxonomy of update in (evolving) information systems is discussed, in the remainder of this chapter, the need for the kinds of updates introduced is discussed by relating them to the state-transition model presented above. On this basis, three kinds of update are distinguished: *events*, *recording* and *correction* ([FOP92a], [FOP92c]). These kinds of update are related to each other in a formal framework for updates in evolving information systems, within this framework an appropriate partial algebra is defined, for each kind of update. The notion of *forgetting* is not addressed in this chapter, as it introduces a whole new range of problems (see subsection 9.2.1.1).

4.2 The Event Level

An event taking place in a universe of discourse, usually a part of an organisation, is considered to occur on the *organisational level* ([FOP92a]). See also figure 4.2. The corresponding event in the information system, implying transitions at application model versions, is considered to occur at the so-called *occurrence level*. In the previous chapter, we stated that an application model history is (the result) of a sequence of events (which are observed to be) occurring in the universe of discourse. This is embodied in the relations Behaves and IsEIS, which correspond to the homomorphism denoted in figure 4.1 (page 60).

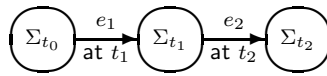


Figure 4.3: Application Model History

Figure 4.3, gives a graphical representation of a sample application model history (at the event level) in the state-transition oriented view as derived in the previous section. This example is based on the sequence of events observed in the universe of discourse taken from figure 4.2. For a given application model universe \mathcal{U}_Σ the partial algebra for the specification of updates on the *event level* is:

$$\mathcal{A}_{ev} \triangleq \langle \mathcal{AMH}, \text{Occ}, \text{StartAMH} \rangle$$

The first operation (Occ) defines the result of performing an event occurrence on a given application model history, and is identified by:

Definition 4.2.1

$$\begin{aligned}
 &\text{Occ} : \mathcal{M} \times \mathcal{T} \times \mathcal{AMH} \mapsto \mathcal{AMH} \\
 &\text{Occ}(m, t, H) \triangleq \text{if } \exists!_{H'} [H \llbracket m \rrbracket_t H'] \text{ then} \\
 &\quad \text{that } H' \\
 &\quad \text{else} \\
 &\quad \text{error: invalid event occurrence} \\
 &\quad \text{fi}
 \end{aligned}$$

□

The correctness of the algebra with respect to an appropriate well-formedness criterion will be proven for the partial algebra for updates at the event level, and the partial algebras for the two levels to come. As a first step, the following lemma states that, recording the occurrence of an event in an application model history H of which every prefix is a *correct application model history* (IsAMH) yields an application model history with the same property:

Lemma 4.2.1 (invariant: correct application model history)

If $\text{Occ}(m, t, H) = H'$, then:

$$\forall u \in \mathcal{T} [\text{IsAMH}(H|_u)] \Rightarrow \forall u \in \mathcal{T} [\text{IsAMH}(H'|_u)]$$

Proof:

$$\begin{aligned}
\forall_{u \in \mathcal{T}} [\text{IsAMH}(H|_u)] \wedge H' &\neq \text{error} && \Rightarrow \{\text{definition of Occ}\} \\
\forall_{u \in \mathcal{T}} [\text{IsAMH}(H|_u)] \wedge H \llbracket m \rrbracket_t H' &&& \Rightarrow \{\text{lemma 3.5.2 (page 58)}\} \\
\forall_{u \in \mathcal{T}} [\text{IsAMH}(H|_u)] \wedge \text{IsAMH}(H) \wedge H \llbracket m \rrbracket_t H' &&& \Rightarrow \{\text{corollary 3.3.2 (page 47)}\} \\
\forall_{u \in \mathcal{T}} [\text{IsAMH}(H|_u)] \wedge \text{IsAMH}(H) \wedge \forall_{u > t} [\text{IsAMH}(H'_u)] &&& \Rightarrow \{\text{corollary 3.3.4 (page 47)}\} \\
\forall_{u \leq t} [\text{IsAMH}(H'_u)] \wedge \forall_{u > t} [\text{IsAMH}(H'_u)] &&& \Rightarrow \{\text{elaborate}\} \\
\forall_{u \in \mathcal{T}} [\text{IsAMH}(H'_u)] &&&
\end{aligned}$$

□

With the aid of the lemma given above, the following two theorems on the *well-behavedness* of an evolving information system (specified as E, H) can be proven. One, performing an occurrence operation on a *well-behaved evolving information system*, also results in a well-behaved evolving information system:

Theorem 4.2.1 (*invariant: well-behaved evolving information system*)

If $\text{Occ}(m, t, H) = H'$ and $t > \text{Last}(E)$, then:

$$\text{Behaves}(E, H) \Rightarrow \text{Behaves}(E \oplus t:m, H')$$

Note that $t > \text{Last}(E)$ requires t to be later than the last point in time at which E is defined.

Proof:

From $\text{Occ}(m, t, H) = H'$ follows: $H \llbracket m \rrbracket_t H'$. Using this in conjunction with the definition 3.2.4 (page 42) of Behaves and $\llbracket \cdot \rrbracket^{cwa}$, we derive:

$$\begin{aligned}
\forall_{u \in \mathcal{T}} [H|_u \llbracket E \rrbracket_u^{cwa} H|_{\triangleright u}] &&& \Rightarrow \{\text{corollary 3.3.4 (page 47)}\} \\
\forall_{u < t} [H'_u \llbracket E \rrbracket_u^{cwa} H'|_{\triangleright u}] &&& \Rightarrow \{\text{corollary 3.3.3 (page 47)}\} \\
\forall_{u < t} [H'_u \llbracket E \rrbracket_u^{cwa} H'|_{\triangleright u}] \wedge H|_t \llbracket m \rrbracket_t H'|_{\triangleright t} &&& \Rightarrow \{\text{assumption: } t > \text{Last}(E)\} \\
\forall_{u \leq t} [H'_u \llbracket E \oplus t:m \rrbracket_u^{cwa} H'|_{\triangleright u}] &&& \Rightarrow \{\text{corollary 3.3.1 (page 47)}\} \\
\forall_{u \leq t} [H'_u \llbracket E \oplus t:m \rrbracket_u^{cwa} H'|_{\triangleright u}] \wedge \forall_{u > t} [H' = H'_u] &&& \Rightarrow \{\text{definition of } \llbracket \cdot \rrbracket^{cwa}\} \\
\forall_{u \in \mathcal{T}} [H'_u \llbracket E \oplus t:m \rrbracket_u^{cwa} H'|_{\triangleright u}] &&&
\end{aligned}$$

□

Two, the occurrence operation leads to a well-behaved proper evolving information system if it is performed on an evolving information system with the same property:

Theorem 4.2.2 (*invariant: well-behaved proper evolving information system*)

If $\text{Occ}(t, m, H) = H'$ and $t > \text{Last}(E)$, then:

$$\text{IsEIS}(E, H) \Rightarrow \text{IsEIS}(E \oplus t:m, H')$$

Proof:

From theorem 4.2.1 follows:

$$\text{Behaves}(E, H) \Rightarrow \text{Behaves}(E \oplus t:m, \text{Occ}(m, t, H))$$

Further, from lemma 4.2.1 follows:

$$\forall_{u \in \mathcal{T}} [\text{IsAMH}(H|_u)] \Rightarrow \forall_{u \in \mathcal{T}} [\text{IsAMH}(H'_u)]$$

Application of the definition of IsEIS then yields the result. \square

The initial application model history consists of only one empty application model state, and is identified by:

Definition 4.2.2

$$\text{StartAMH} \triangleq \emptyset$$

\square

For this definition we immediately have:

Proposition 4.2.1 (*correct initial application model history*)

$$\forall_{u \in \mathcal{T}} [\text{IsAMH}(\text{StartAMH}|_u)]$$

To express the correctness of the partial algebras for the distinct kinds of updates distinguished, a notion of *closedness* for the operations of the partial algebras is called for. We deem a partial algebra $\langle X, F \rangle$ to be *closed* with respect to a property P , if for all operations $f \in F$:

$$P(x_1) \wedge \dots \wedge P(x_n) \wedge f(x_1, \dots, x_n) \downarrow \Rightarrow P(f(x_1, \dots, x_n))$$

where n is the arity of operation f . From lemma 4.2.1 and proposition 4.2.1 it follows that the partial algebra for the event level is closed with respect to the unary property P :

$$P(H) \triangleq \forall_{u \in \mathcal{T}} [\text{IsAMH}(H|_u)]$$

The closedness with respect to some well-formedness property is also proven for the partial algebras which are defined in the remainder of this chapter,

4.3 The Recording Level

A second level on which state transitions take place, the *recording level*, is introduced in this section. Whenever an event occurs in an organisation, it is communicated to the information system involved, by an update request (see figure 1.11 (page 14)). The processing of this update request, called the recording of an event, results in an appropriate state transition in the information system. As has been argued in e.g. [SA85], [Sno90] and [FOP92c], the time of recording of an event is generally different from its time of occurrence. The time at which an event occurs in a universe of discourse is referred to as the *valid time*, whereas the time at which the event is recorded is called the *transaction time*.

The resulting state transition may be more than a single elementary transition of an application model version. It can be seen as a transition of the complete application model history, modelling the history of the organisation up to the occurrence of the newly recorded event, to a new history. A sequence of these application model history transitions, produced by successive recordings, is called an *application model recording history*. Such an application

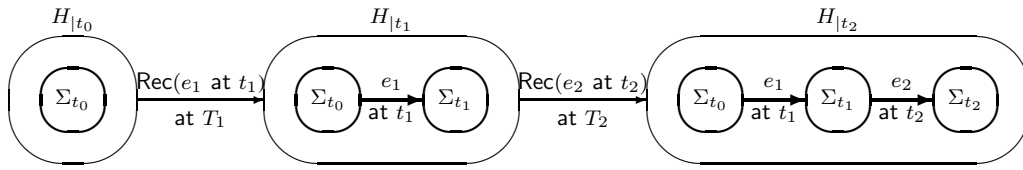


Figure 4.4: Application Model Recording History

model recording history reflects both the events occurring in the organisation, and the recording of these events in the information system.

In figure 4.4, the graphical representation of application model version transitions on the event level is extended by the recording level at which transitions of the application model history take place, due to the recording of these events. Note that the t_i 's are valid points in time, while the T_i 's are transaction points in time.

The example application model recording history, in figure 4.4, corresponds to the successive recording of events e_1 and e_2 observed by the user in figure 4.2 (page 61). The arrows, representing transitions between application model histories, are labeled with the denotation of the recording of the event causing the transition, and the transaction time of the event in question. The application model histories contained in the recording history expand over the course of time, the older ones being a prefix of the newer ones. An application model recording history can thus be defined formally as a triple $\langle R, E, H \rangle$, where H is the complete application model history, E is the sequence of *event occurrences*, and R is the mapping from the transaction time to the valid time of the recorded events. The domain of application model recording histories is thus identified by:

Definition 4.3.1

$$\mathcal{AMRH} \triangleq (\mathcal{T} \rightarrow \mathcal{T}) \times \mathcal{EO} \times \mathcal{AMH}$$

□

Note that $E \cdot R$ maps the transaction time of an event to an action $a \in \mathcal{M}$. The associated partial algebra for the recording of event occurrences is:

$$\mathcal{A}_{rec} \triangleq \langle \mathcal{AMRH}, \text{Rec}, \text{StartAMRH} \rangle$$

The Rec operation defines the result of recording an event occurrence on a given application model recording history $\langle R, E, H \rangle$. Since recordings should be done monotonously, the time of recording (t_r) of an event occurrence should be later than the last recording in R , so: $t_r > \text{Last}(E \cdot R)$. Further, the time at which the event occurs should be later than any of the recorded events in E , so: $t_e > \text{Last}(E)$. A violation of this latter condition will require correction (see next section). If the transaction time and the valid time are correct, the new application model recording history consists of the old one, extended by the application model history resulting from applying the (to be recorded) event occurrence.

Definition 4.3.2

$\text{Rec} : \mathcal{M} \times \mathcal{T} \times \mathcal{T} \times \mathcal{AMRH} \rightarrow \mathcal{AMRH}$

$\text{Rec}(m, t_e, t_r, \langle R, E, H \rangle) \triangleq$ **if** $t_r > \text{Last}(E \cdot R)$ **then**

if $t_e > \text{Last}(E) \wedge \text{Occ}(m, t_e, H) \neq \text{error}$ **then**

$\langle R \oplus t_r : t_e, E \oplus t_e : m, \text{Occ}(m, t_e, H) \rangle$

else

error: late recording

fi

else

error: illegal transaction time

fi

□

We can define correctness of application model recording histories in an analogous manner to the correctness of an evolving information system; for such an application model recording history $\langle R, E, H \rangle$ to be correct, E and H must be a well-behaved proper evolving information system, and R must record all events in E in the proper order. Note that, for the moment, we assume the ideal situation in which events are recorded in their order of occurrence. In the next section this assumption is relaxed to some extent. The formal definition of a correct application model recording history is:

Definition 4.3.3

$$\begin{aligned} \text{IsAMRH}(R, E, H) &\triangleq \text{IsEIS}(E, H) \wedge \text{dom}(E) = \text{ran}(R) \\ &\wedge \forall_{t_1, t_2 \in \text{dom}(R)} [t_1 < t_2 \Rightarrow R(t_1) < R(t_2)] \end{aligned}$$

□

The following corollary is a direct result of this definition, that states that R indeed preserves the order of transaction times:

Corollary 4.3.1 If $\text{IsAMRH}(R, E, H)$ we have:

$$\forall_{t_1, t_2 \in \text{dom}(R)} [t_1 < t_2 \iff R(t_1) < R(t_2)]$$

Proof:

From the definition of IsAMRH immediately follows:

$$\forall_{t_1, t_2 \in \text{dom}(R)} [t_1 < t_2 \Rightarrow R(t_1) < R(t_2)]$$

As a result, we only have to prove:

$$\forall_{t_1, t_2 \in \text{dom}(R)} [R(t_1) < R(t_2) \Rightarrow t_1 < t_2]$$

If $R(t_1) < R(t_2)$.

Since R is a function, it cannot be had that $t_1 = t_2$.

For $t_2 < t_1$ we have $R(t_2) < R(t_1)$, leading to a contradiction.

Therefore $t_1 < t_2$. □

A further result of the above definition, is that R is a bijection from $\text{dom}(R)$ to $\text{ran}(R)$. The first invariant on the Rec operation states that the recording of a (correct) event occurrence on a well-behaved, proper evolving, information system, results in a well-behaved, proper evolving, information system:

Theorem 4.3.1 (*invariant: well-behaved proper evolving information system*)

If $\text{Rec}(m, t_e, t_r, \langle R, E, H \rangle) = \langle R', E', H' \rangle$ then:

$$\text{IsEIS}(E, H) \Rightarrow \text{IsEIS}(E', H')$$

Proof:

$$\text{IsEIS}(E, H) \wedge \text{Rec}(m, t_e, t_r, \langle R, E, H \rangle) \neq \text{error} \quad \Rightarrow \quad \{\text{definition of Rec}\}$$

$$\text{IsEIS}(E, H) \wedge t_e > \text{Last}(E) \wedge \text{Occ}(t_e, m, H) \neq \text{error} \quad \Rightarrow \quad \{\text{theorem 4.2.2 (page 63)}\}$$

$$\text{IsEIS}(E \oplus t:m, \text{Occ}(t_e, m, H)) \quad \Rightarrow \quad \{\text{definition of Rec}\}$$

$$\text{IsEIS}(E', H')$$

□

The definition of a correct application model history (IsAMRH) allows us to state a second invariant on the Rec operation. The invariant states that applying a (correct) recording of an event to a correct application model recording history results in a correct application model recording history.

Theorem 4.3.2 (*invariant: correct application model history*)

If $\text{Rec}(m, t_e, t_r, \langle R, E, H \rangle) = \langle R', E', H' \rangle$ then:

$$\text{IsAMRH}(R, E, H) \Rightarrow \text{IsAMRH}(R', E', H')$$

Proof:

As the definition of IsAMRH is a conjunction of three parts, the proof can also be split in three parts:

1. From theorem 4.3.1 follows directly: $\text{IsEIS}(E, H) \Rightarrow \text{IsEIS}(E', H')$.
2. As $\text{Rec}(m, t_e, t_r, \langle R, E, H \rangle)$ we have: $t_r > \text{Last}(E \cdot R)$ and $t_e > \text{Last}(E)$. Further, as $\text{IsAMRH}(R, E, H)$ also: $\text{dom}(E) = \text{ran}(R)$. From this we derive:

$$\begin{aligned} t_r > \text{Last}(E \cdot R) \quad \wedge \quad t_e > \text{Last}(E) &\Rightarrow \{ \text{definition of Last} \} \\ t_r > \max \cdot \text{dom}(E \cdot R) \quad \wedge \quad t_e > \max \cdot \text{dom}(E) &\Rightarrow \{ \text{dom}(E) = \text{ran}(R) \} \\ t_r > \max \cdot \text{dom}(R) \quad \wedge \quad t_e > \max \cdot \text{ran}(R) & \end{aligned}$$

Finally, applying this result, to: $\forall_{t_1, t_2 \in \text{dom}(R)} [t_1 < t_2 \Rightarrow R(t_1) < R(t_2)]$ yields:

$$\forall_{t_1, t_2 \in \text{dom}(R')} [t_1 < t_2 \Rightarrow R'(t_1) < R'(t_2)]$$

3. From the definition of Rec follows:

$$\text{dom}(E') = \text{dom}(E) \oplus t_e \wedge \text{ran}(R') = \text{ran}(R) \oplus t_e$$

As $\text{dom}(E) = \text{ran}(R)$, we have: $\text{dom}(E') = \text{ran}(R')$.

□

Initially, an application model recording history consists of only one (possibly empty) application model history. This initial application model history (containing the initial application model state), is identified by:

Definition 4.3.4

$$\text{StartAMRH} \triangleq \langle \emptyset, \emptyset, \text{StartAMH} \rangle$$

□

As a direct result of this definition, we also have:

Corollary 4.3.2

$$\text{IsAMRH}(\text{StartAMRH})$$

Proof:

Follows directly from proposition 4.2.1 (page 64), and the definition of IsAMRH.

□

From theorem 4.3.2, and corollary 4.3.2, it follows that the Rec and StartAMRH operations are closed with respect to the IsAMRH invariant.

4.4 The Correction Level

In an ideal (evolving) information system, the application model history reflects the history of the modelled universe of discourse in a(n) (empirically) correct way at any point in time. It can, indeed, be checked whether the application model is consistent with the meta model, but there is no automatable way to guarantee that the application model history reflects the history of the universe of discourse as it actually occurred. In terms of figure 4.1, the function h must be validated empirically. Any discrepancy between the state transitions in the universe of discourse, and those modelled in the application model, cannot be detected automatically. This implies that in the case of a flaw found in the history of the application model, an update needs to be performed to *correct* this flaw, i.e. correcting the mapping h . This kind of update is called a *correction*, and corrects recordings which have already been performed, or which have erroneously, not been performed at all.

A remaining problem is that the organisation may have made decisions based on the corrected information, that may have to be reconsidered. A proper correction of the information system may turn out to be impossible because of the long term effects of these (in retrospect erroneously) decisions. For instance, a purchase order for new supplies may have been sent out automatically, while the stock supply was in reality sufficient, this situation cannot be restored by simply correcting the erroneously recorded event occurrence.

The correction operations presented here do not support the analysis of decisions made on the corrected information, such support can only be provided when a record is maintained of all queries ever performed on the stored information, and this is re-evaluated after the correction. The information about corrections, however, is stored in the information system, and can therefore be queried by the users, as a result, *post correction* analysis can be performed, and further recordings can be made to cope with the long term effects of the erroneous decisions.

4.4.1 A Taxonomy of Correction

In this subsection we provide a taxonomy of the sorts of correction needed. As stated in section 4.1, an information system reflects an organisation correctly if and only if there exists an epimorphism between the states and transitions of the application model and the states and transitions in the universe of discourse. From this requirement it follows that the order in which the events occurred in the universe of discourse (should) be the same as the order of the recorded events.

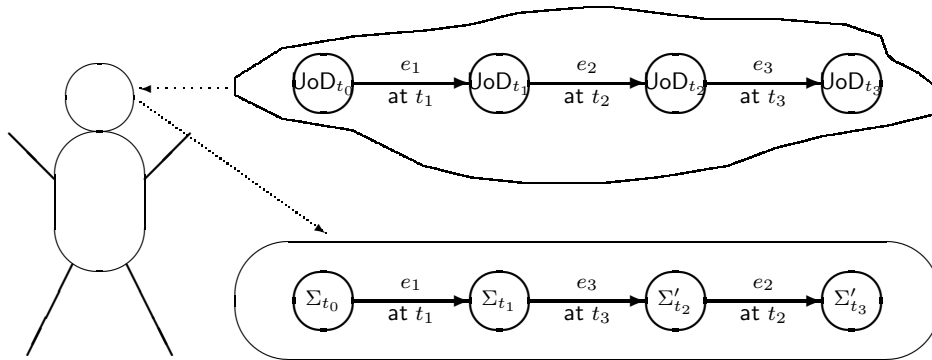


Figure 4.5: Recording events in the wrong order

Example 4.4.1

As an illustration of the undesired effects of recording events in the wrong order, consider the following two events for the CD store example in chapter 1.

- e_1 : ADD Song: 'Walk of Life' recorded on Record: 'Brothers in Arms'
- e_2 : DELETE Song recorded on Record: 'Brothers in Arms'

The first event specification (e_1) represents the addition of the recording of 'Walk of Life' on Record 'Brothers in Arms', whereas the second event specification (e_2) represents the deletion of all songs recorded on Record 'Brothers in Arms'. Note that the latter event specification is an intentional specification of a set of events.

When recording e_1 , e_2 in the obvious order (e_1 followed by e_2), a state of the system results in which the Record 'Brothers in Arms' contains no songs. Conversely, when e_2 is recorded before e_1 , i.e. in the wrong order, the result will be that the song 'Walk of Life' is the only song recorded on the Record 'Brothers in arms'. \square

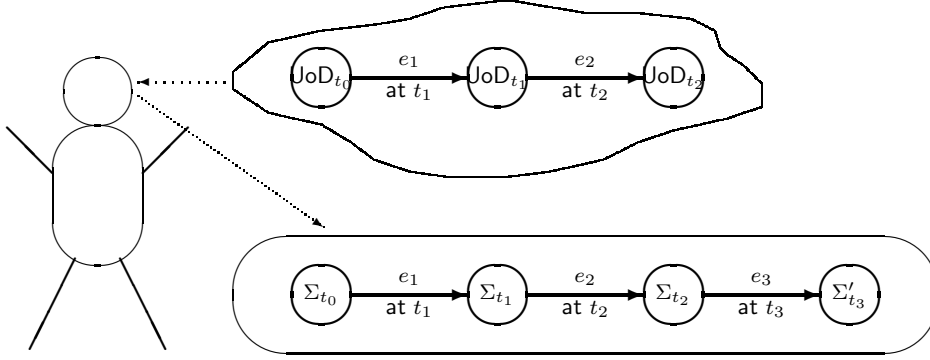


Figure 4.6: Recording of ghost events

From the above example can be concluded that events should be recorded correctly, in the order of their occurrence. In practice, events may indeed be recorded out of order, e.g. recordings may be performed too late. Such a late recording causes the order in which the events occurred according to the information system to be different from the order in which they actually occurred in the universe of discourse. Further, an event which did not actually happen at all, or occurred differently, may have been recorded. These situations are illustrated in figures 4.5, 4.6 and 4.7 respectively. This leads to the identification of three kinds of correction: the insertion of a (late) recording of an event in the sequence of recordings already performed, the removal of a recording already performed, and the replacement of a recording by a new recording of another event.

To accomplish the kinds of corrections identified, it must be possible to *travel* backwards in the sequence of recordings. The operation which accomplishes this task is called a *rewind*, and is the most primitive form of correction. In [OPF92], it is shown that the three kinds of corrections discussed above can be mapped onto rewinds and re-recordings. Therefore, we define the semantics of all correction operations in terms of a rewind and a re-recording of a sequence of events.

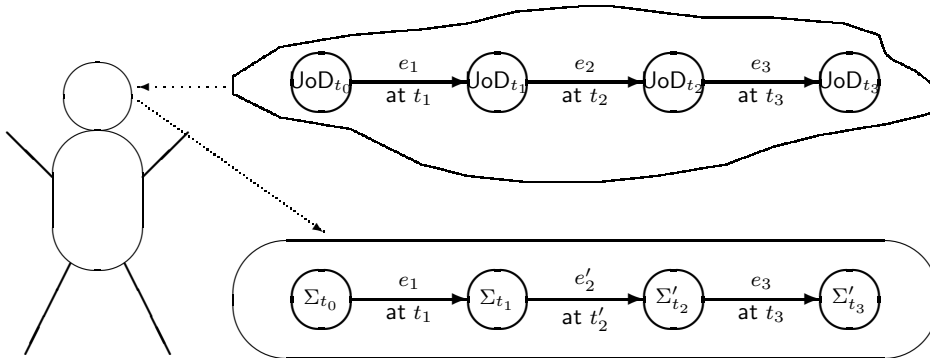


Figure 4.7: Recording of wrong event

The minimal set of operations needed to perform corrections and recordings (rewind and (re-)recording) is referred to as the *primitive update requests*. The more extensive set of correction/recording operators (remove, replace, insert, record, rewind) is referred to as the (user) *update requests*. The partial algebras for the primitive and user update requests are defined in the next two subsections. The resulting levels of update requests are illustrated in figure 4.8.

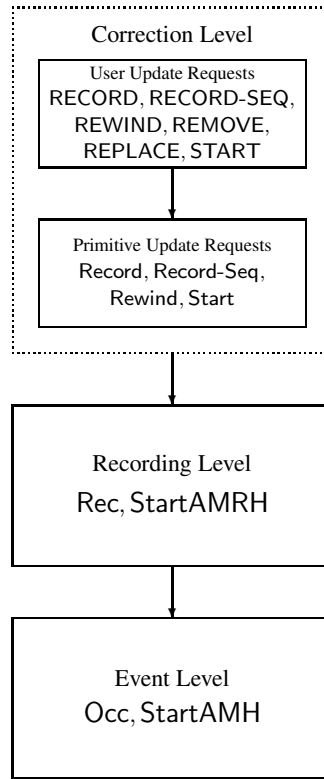


Figure 4.8: Levels of update requests

4.4.2 Update Requests on a Primitive Level

Figure 4.9 represents, graphically, the performance of a correction using a *rewind* for the case of a *replacement* of the recording of an event e_1 occurring at t_1 by a recording of event e'_1 . The replacement is assumed to take place after the recording of event e_2 . The invalidated application model histories $H|_{t_1}$ and $H|_{t_2}$ are not forgotten. This means that incorrect situations can still be retrieved to reconstruct the information used in past decisions (based on information which has now been proven to be wrong).

A sequence of successive recordings, an application model recording history, can be seen as the belief of the universe of discourse by the information system. This means that a correction of this belief of the world is performed by a rewind causing a transition of the current application model recording history in the information system to a new one without forgetting the old one. A sequence of these application model recording history transitions produced by rewinds is called the *application model evolution*:

Definition 4.4.1

$$AMEV \triangleq \mathcal{T} \mapsto AMRH$$

□

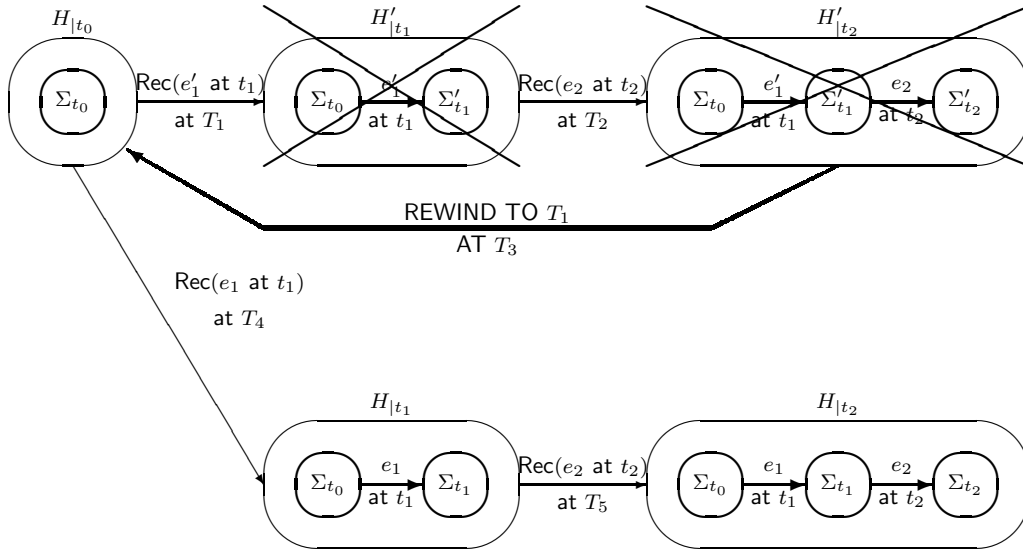


Figure 4.9: Correction by a rewind

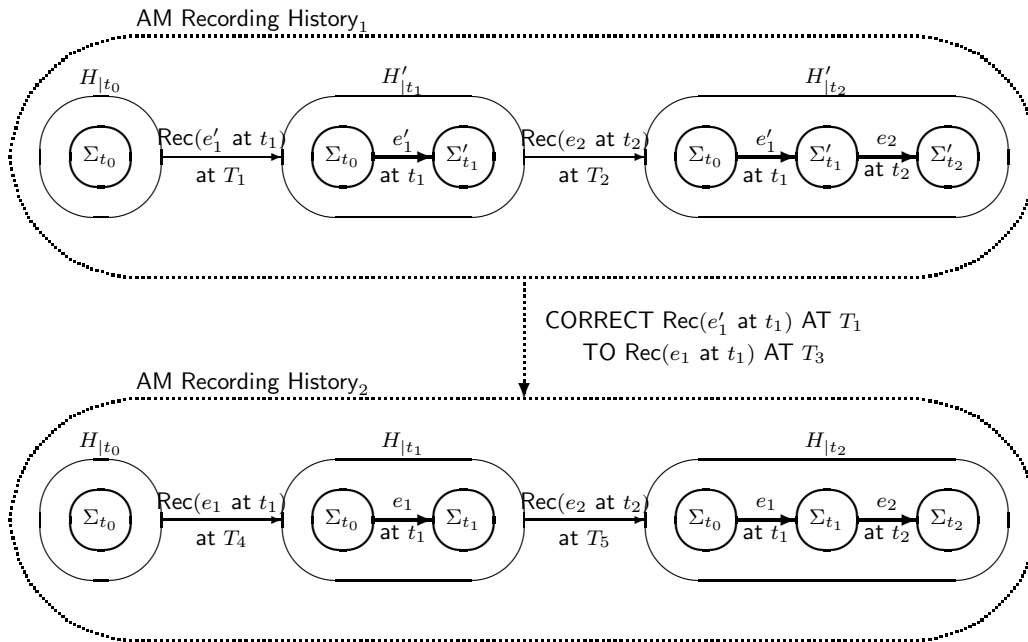


Figure 4.10: Application Model Evolution

In figure 4.10 the situation of figure 4.9 is represented in an alternative way that identifies the three levels of state transitions more clearly. Note that the rewind performed by the correction is present implicitly in this figure. Corrections requiring the removal or insertion of a recording of an event can be represented in the same way.

The transitions discussed above are considered to take place on what is termed the *correction level*. For this level, we define two algebras. The first algebra is the *primitive level* containing only recording (of both single events and sequences of events) and rewind as operations. The second algebra, which is defined in the next subsection, defines a more convenient set of operations on application model evolutions. The partial algebra at the primitive level is:

$$\mathcal{A}_{prim} \triangleq \langle \mathcal{AM\mathcal{EV}}, \text{Rewind}, \text{Record}, \text{Record-Seq}, \text{Start} \rangle$$

A decrement operator for points in time is needed for the definition of the operators of the algebra of the primitive level. This operator is defined in terms of the increment operator as:

Definition 4.4.2 (*decrement*)

$$\triangleleft t \triangleq t' \text{ such that } t' = \triangleright t$$

□

The recording operation for this algebra simply *replaces* the last application model recording history in the application model evolution by the one that results after recording the event on this last application model history.

Definition 4.4.3

$$\begin{aligned} \text{Record} : \mathcal{M} \times \mathcal{T} \times \mathcal{T} \times \mathcal{AMEV} &\mapsto \mathcal{AMEV} \\ \text{Record}(m, t_e, t_r, V) &\triangleq \text{if } \text{Rec}(m, t_e, t_r, V \cdot \text{Last}(V)) \neq \text{error} \text{ then} \\ &\quad V \oslash \text{Last}(V) : \text{Rec}(m, t_e, t_r, V \cdot \text{Last}(V)) \\ &\quad \text{else} \\ &\quad \text{error} \\ &\quad \text{fi} \end{aligned}$$

Note that Last(V) returns the last point in time at which V is defined (see appendix A).

□

An application model evolution is considered to be correct, if all application model recording histories contained in it are correct:

Definition 4.4.4

$$\text{IsAMEV}(V) \triangleq V \downarrow t \Rightarrow \text{IsAMRH}(V(t))$$

□

The Record operation maintains the correctness of application model evolutions:

Theorem 4.4.1 If $\text{Record}(m, t_e, t_r, V) = V'$, then:

$$\text{IsAMEV}(V) \Rightarrow \text{IsAMEV}(V')$$

Proof:

From the definition of Rec it follows that V and V' only differ for their values at $\text{Last}(V)$, and furthermore that $V' \cdot \text{Last}(V) = \text{Rec}(m, t_e, t_r, V \cdot \text{Last}(V))$.

As $\text{IsAMEV}(V)$, we have $\text{IsAMRH}(V \cdot \text{Last}(V))$.

From theorem 4.3.2 (page 67) it follows that $\text{IsAMRH}(\text{Rec}(m, t_e, t_r, V \cdot \text{Last}(V)))$, and thus we have $\text{IsAMEV}(V')$.

□

The recording of event can be generalised to the recording of a whole sequence of event occurrences:

Definition 4.4.5

$$\begin{aligned} \text{Record-Seq} : \mathcal{EO} \times \mathcal{T} \times \mathcal{AMEV} &\mapsto \mathcal{AMEV} \\ \text{Record-Seq}(E, t_r, V) &\triangleq \text{if } E \neq \emptyset \text{ then} \\ &\quad \text{let } t_e = \text{First}(E), m = E(t_e); \\ &\quad \text{if } \text{Record}(m, t_e, t_r, V) \neq \text{error} \text{ then} \\ &\quad \quad \text{Record-Seq}(E \ominus t_e : m, \triangleright t_r, \text{Record}(m, t_e, t_r, V)) \\ &\quad \text{else} \\ &\quad \quad \text{error} \\ &\quad \text{fi} \\ &\quad \text{else } V \text{ fi} \end{aligned}$$

□

As Record-Seq is defined directly in terms of Record, we immediately have:

Corollary 4.4.1 If $\text{Record-Seq}(E, t_r, V) = V'$, then:

$$\text{IsAMEV}(V) \Rightarrow \text{IsAMEV}(V')$$

A rewind to a given event time, can only be recorded after the last recording or rewind in V , further, the event time must be of an event which indeed occurred. The performance of a rewind, to a given point in time (t_b), means that an application model recording history must be added, that gives all recordings of events which occurred after the point in time (t_b). The effect of performing a rewind on an application model element evolution is identified as:

Definition 4.4.6

Rewind : $\mathcal{T} \times \mathcal{T} \times \mathcal{AMEV} \mapsto \mathcal{AMEV}$

Rewind(t_b, t_r, V) \triangleq **let** $\langle R, E, H \rangle = V \cdot \text{Last}(V)$;
 if $t_r > \text{Last}(V) \wedge t_r > \text{Last}(R)$ **then**
 if $E \downarrow t_b$ **then**
 $V \oplus t_r : \langle (R_{< t_b}^{\leftarrow})^{\leftarrow}, E_{< t_b}, H_{| t_b} \rangle$
 else
 error: undefined recording
 fi
 else
 error: late rewind
 fi

where $X_{< t} \triangleq \lambda s. \text{if } s < t \text{ then } X(s) \text{ else } \perp$ **fi**

Note that the inverse R^{\leftarrow} of R is only defined if R is a bijection. If $\text{IsAMRH}(R, E, H)$, this is indeed the case (see corollary 4.3.1 (page 66)). \square

The first step towards the proof of the closedness of the Rewind operation with respect to IsAMRH , is the following lemma stating that the suffix of a prefixed evolving information system $(E_{< t}, H_{| t})$, is well-behaved:

Lemma 4.4.1 If $H \in \mathcal{AMH}$ and $t \in \mathcal{T}$, then:

$$\forall t' \geq t \left[(H_{| t})_{| t'} \ll^{cwa} (H_{| t})_{| \triangleright t'} \right]$$

Proof:

The proof follows from two observations:

1. From the definition of $E_{< t}$ follows: $\forall t' \geq t \left[\neg E_{< t} \downarrow t' \right]$.
2. From corollary 3.2.1 (page 42) follows: $\forall t' \geq t \left[(H_{| t})_{| t'} = H_{| t} \right]$.

Then, the definition 3.2.4 (page 42) of \ll^{cwa} gives the result. \square

As a next step, we prove that the application model recording history resulting from a rewind is a valid one:

Lemma 4.4.2 If $\text{Rewind}(t_e, t_r, V) = V'$, then:

$$\text{IsAMEV}(V) \Rightarrow \text{IsAMRH}(V'(t_r))$$

Proof:

From the definition of IsAMEV , and $\text{IsAMEV}(V)$ it follows directly that:

$$\text{IsAMRH}(V \cdot \text{Last}(V))$$

so in particular: $\text{IsAMRH}(R, E, H)$ where $\langle R, E, H \rangle = V \cdot \text{Last}(V)$.

So we have to prove: $\text{IsAMRH}((R_{<t_b}^{\leftarrow})^{\leftarrow}, E_{<t_b}, H_{|t_b})$.

When substituting the definition of Behaves, IsAMH, IsEIS in the definition of IsAMRH we get:

$$\begin{aligned} \text{IsAMRH}(R, E, H) &\iff \forall_{t \in \mathcal{T}} [H_{|t} \llbracket E \rrbracket_t^{cwa} H_{| \triangleright t}] \\ &\quad \wedge \quad \forall_{t \in \mathcal{T}} [\text{IsAMH}(H_{|t})] \\ &\quad \wedge \quad \text{dom}(E) = \text{ran}(R) \\ &\quad \wedge \quad \forall_{t_1, t_2 \in \text{dom}(R)} [t_1 < t_2 \Rightarrow R(t_1) < R(t_2)] \end{aligned}$$

As a result, the proof can also be split into four parts:

1. $\forall_{t \in \mathcal{T}} [H_{|t} \llbracket E \rrbracket_t^{cwa} H_{| \triangleright t}] \quad \Rightarrow \{elaborate\}$
 $\forall_{t < t_b} [H_{|t} \llbracket E \rrbracket_t^{cwa} H_{| \triangleright t}] \quad \Rightarrow \{definition\ of\ E_{<t_b}\}$
 $\forall_{t < t_b} [H_{|t} \llbracket E_{<t_b} \rrbracket_t^{cwa} H_{| \triangleright t}] \quad \Rightarrow \{corollary\ 3.2.1\ (page\ 42)\}$
 $\forall_{t < t_b} [(H_{|t_b})_{|t} \llbracket E_{<t_b} \rrbracket_t^{cwa} (H_{|t_b})_{| \triangleright t}] \quad \Rightarrow \{lemma\ 4.4.1\ (page\ 73)\}$
 $\forall_{t \in \mathcal{T}} [(H_{|t_b})_{|t} \llbracket E_{<t_b} \rrbracket_t^{cwa} (H_{|t_b})_{| \triangleright t}]$
2. If $\forall_{t \in \mathcal{T}} [\text{IsAMH}(H_{|t})]$, then also $\forall_{t \in \mathcal{T}} [\text{IsAMH}((H_{|t_b})_{|t})]$.
3. As $\text{IsAMRH}(R, E, H)$, we have: $\text{ran}(R) = \text{dom}(E)$.

Using this, we derive:

$$\begin{aligned} \text{ran}((R_{<t_b}^{\leftarrow})^{\leftarrow}) &= \{definition\ of\ (R_{<t_b}^{\leftarrow})^{\leftarrow}\} \\ \{s \mid s \in \text{dom}(R^{\leftarrow}) \wedge s < t_b\} &= \{definition\ of\ \text{dom}(R^{\leftarrow})\} \\ \{s \mid s \in \text{ran}(R) \wedge s < t_b\} &= \{\text{ran}(R) = \text{dom}(E)\} \\ \{s \mid s \in \text{dom}(E) \wedge s < t_b\} &= \{definition\ of\ \text{dom}\} \\ \text{dom}(E_{<t_b}) & \end{aligned}$$

4. We already have: $(R_{<t_b}^{\leftarrow})^{\leftarrow} \subseteq R$.

If: $\forall_{t_1, t_2 \in \text{dom}(R)} [t_1 < t_2 \Rightarrow R(t_1) < R(t_2)]$, we also have:

$$\forall_{t_1, t_2 \in \text{dom}((R_{<t_b}^{\leftarrow})^{\leftarrow})} [t_1 < t_2 \Rightarrow (R_{<t_b}^{\leftarrow})^{\leftarrow}(t_1) < (R_{<t_b}^{\leftarrow})^{\leftarrow}(t_2)]$$

□

Using the lemma given above, we can formulate the closedness with respect to the IsAMEV property of the Rewind operation as follows:

Theorem 4.4.2 If $\text{Rewind}(t_e, t_r, V) = V'$, then:

$$\text{IsAMEV}(V) \Rightarrow \text{IsAMEV}(V')$$

Proof:

$$\begin{aligned}
\text{IsAMEV}(V) &\Rightarrow \{\text{lemma 4.4.2 (page 73)}\} \\
\text{IsAMEV}(V) \wedge \text{IsAMRH}(V'(t_r)) &\Rightarrow \{\text{definition of Rewind}\} \\
\text{IsAMEV}(V) \wedge \text{IsAMRH}(V'(t_r)) \wedge \\
\forall_{t \in \text{dom}(V)} [V(t) = V'(t)] &\Rightarrow \{\text{definition of IsAMEV}\} \\
\text{IsAMRH}(V'(t_r)) \wedge \\
\forall_{t \in \text{dom}(V)} [\text{IsAMRH}(V(t)) \wedge V(t) = V'(t)] &\Rightarrow \{\text{elaborate}\} \\
\forall_{t \in \text{dom}(V) \oplus t_r} [\text{IsAMRH}(V'(t))] &\Rightarrow \{\text{definition of Rewind}\} \\
\forall_{t \in \text{dom}(V')} [\text{IsAMRH}(V'(t))] &\Rightarrow \{\text{definition of IsAMEV}\} \\
\text{IsAMEV}(V') &
\end{aligned}$$

□

There also exists an initialisation operation for the partial algebra on this level,

Definition 4.4.7

$$\begin{aligned}
\text{Start} : \mathcal{T} &\rightarrow \mathcal{AMEV} \\
\text{Start}(t) &\triangleq \{t : \text{StartAMRH}\}
\end{aligned}$$

□

As $\text{IsAMRH}(\text{StartAMRH})$, the initialisation operation is closed with respect to the IsAMEV property:

Corollary 4.4.2 $\text{IsAMEV}(\text{Start}(t))$

From theorem 4.4.1 (page 72), corollary 4.4.1 (page 73), theorem 4.4.2 (page 74), and corollary 4.4.2 (page 75) it follows that $\mathcal{A}_{\text{prim}}$ is a closed partial algebra with respect to IsAMEV .

4.4.3 Update Requests on the User Level

The partial algebra for updates on the user update level, contains much more convenient operations than the algebra on the primitive update level. These operations are defined in terms of the operations of the partial algebra of the primitive update level, and by some extra (access) functions on application model evolutions. As a result, the closedness with respect to IsAMEV of the algebra for the user update level follows directly from the closedness of the partial algebra on the primitive update level. The partial algebra on the user update level is defined as:

$$\mathcal{A}_{\text{user}} \triangleq \langle \mathcal{AMEV}, \text{REWIND}, \text{RECORD}, \text{RECORD-SEQ}, \text{REMOVE}, \text{REPLACE}, \text{START} \rangle$$

Note that this is just a set of example operations for the user update level. One can always add other ‘exotic’ operations to this algebra.

The rewind operation at this level, is simply defined as the rewind operation at the primitive update level.

Definition 4.4.8

$$\begin{aligned}
\text{REWIND} : \mathcal{T} \times \mathcal{T} \times \mathcal{AMEV} &\rightarrow \mathcal{AMEV} \\
\text{REWIND}(t_b, t_r, V) &\triangleq \text{Rewind}(t_b, t_r, V)
\end{aligned}$$

□

The recording operation at this level, is a more convenient version of the recording operation at the primitive update level. This operation detects when it is dealing with a late-recording, and takes an appropriate action, i.e. a rewind and re-recording. The operation is defined as:

Definition 4.4.9

$$\begin{aligned} \text{RECORD} : \mathcal{M} \times \mathcal{T} \times \mathcal{T} \times \mathcal{AM\mathcal{EV}} &\rightarrow \mathcal{AM\mathcal{EV}} \\ \text{RECORD}(m, t_e, t_r, V) &\triangleq \text{let } \langle R, E, H \rangle = V \cdot \text{Last}(V); \\ &\quad \text{if } E_{t_e \leq} = \emptyset \text{ then} \\ &\quad \quad \text{Record}(m, t_e, t_r) \\ &\quad \text{else} \\ &\quad \quad \text{let } t_b = \text{First}(E_{t_e \leq}); \\ &\quad \quad \text{if } \text{Rewind}(t_b, t_r, V) \neq \text{error} \text{ then} \\ &\quad \quad \quad \text{Record-Seq}(E_{t_e \leq} \oplus t_e : m, \triangleright t_r, \text{Rewind}(t_b, t_r, V)) \\ &\quad \quad \text{else} \\ &\quad \quad \quad \text{error} \\ &\quad \quad \text{fi} \\ &\quad \text{fi} \end{aligned}$$

where $E_{t \leq} \triangleq E \setminus E_{< t}$. □

We can also define an operation to perform a sequence of recordings on the user update level, in a manner analogously to the Record-Seq operation at the primitive update level.

Definition 4.4.10

$$\begin{aligned} \text{RECORD-SEQ} : \mathcal{EO} \times \mathcal{T} \times \mathcal{AM\mathcal{EV}} &\rightarrow \mathcal{AM\mathcal{EV}} \\ \text{RECORD-SEQ}(E, t_r, V) &\triangleq \text{if } E \neq \emptyset \text{ then} \\ &\quad \text{let } t_e = \text{First}(E), m = E(t); \\ &\quad \text{if } \text{RECORD}(m, t_e, t_r, V) \neq \text{error} \text{ then} \\ &\quad \quad \text{RECORD-SEQ}(E \ominus t_e : m, \triangleright t_r, \text{RECORD}(m, t_e, t_r, V)) \\ &\quad \text{else} \\ &\quad \quad \text{error} \\ &\quad \text{fi} \\ &\text{else } V \text{ fi} \end{aligned}$$

□

The next operation introduced, is the removal of a recording which happened at a given point in time. To do this, a rewind must first be performed on the set of recordings, then, the undone recordings (except for the one to be removed) must be re-recorded. This leads to the following definition of the remove operation:

Definition 4.4.11

$$\begin{aligned} \text{REMOVE} : \mathcal{T} \times \mathcal{T} \times \mathcal{AM\mathcal{EV}} &\rightarrow \mathcal{AM\mathcal{EV}} \\ \text{REMOVE}(t_b, t_r, V) &\triangleq \text{let } \langle R, E, H \rangle = V \cdot \text{Last}(V); \\ &\quad \text{if } \text{Rewind}(t_b, t_r, V) \neq \text{error} \text{ then} \\ &\quad \quad \text{Record-Seq}(E_{t_b <}, \triangleright t_r, \text{Rewind}(t_b, t_r, V)) \\ &\quad \text{else} \\ &\quad \quad \text{error} \\ &\quad \text{fi} \end{aligned}$$

□

As well as removing recordings, one also wants to be able to replace a recording with a corrected one. This is done by the replace operation. Replacing an old recording by a new one, is similar to removing a recording, except that the new recording must be recorded. The operation is introduced as:

Definition 4.4.12

$$\begin{aligned} \text{REPLACE} : \mathcal{T} \times \mathcal{M} \times \mathcal{T} \times \mathcal{AMEV} &\rightarrow \mathcal{AMEV} \\ \text{REPLACE}(t_b, m, t_r, V) &\triangleq \text{let } \langle R, E, H \rangle = V \cdot \text{Last}(V); \\ &\quad \text{if } \text{Rewind}(t_b, t_r, V) \neq \text{error} \text{ then} \\ &\quad \quad \text{Record-Seq}(E_{t_b < \oplus t_b : m, \triangleright t_r}, \text{Rewind}(t_b, t_r, V)) \\ &\quad \text{else} \\ &\quad \quad \text{error} \\ &\quad \text{fi} \end{aligned}$$

□

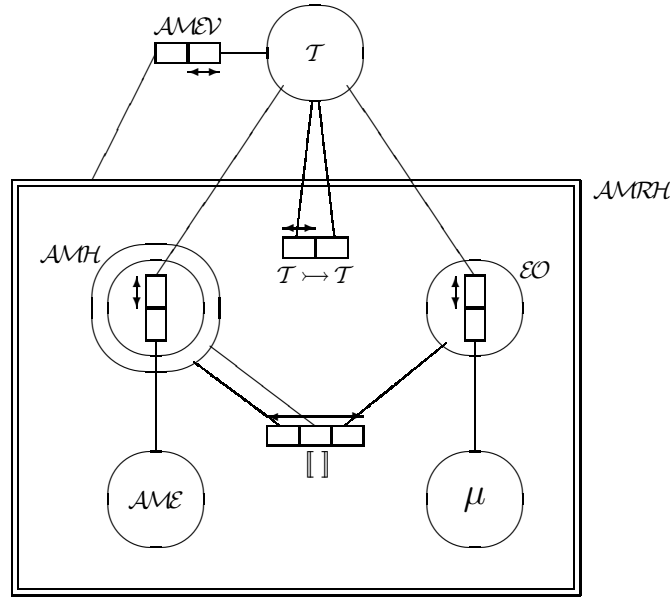


Figure 4.11: Meta schema for update in evolving information systems

The initialisation operation of the algebra for the user update level, is the same as for the primitive update level.

Definition 4.4.13

$$\begin{aligned} \text{START} : \mathcal{T} &\rightarrow \mathcal{AMEV} \\ \text{START}(t) &\triangleq \text{Start}(t) \end{aligned}$$

□

As stated before, the closedness of this partial algebra with respect to IsAMEV follows directly from the primitive update level, since every result of the operations at this level is expressed directly in terms of operations at the primitive level

4.5 Overview of the Framework

In this chapter we provided a taxonomy for update in evolving information systems. As the taxonomy is not based on details of the underlying corpus evolutionis, the taxonomy presented is equally applicable to traditional information systems.

Three levels of updates have been considered, covering the event level, recording level and correction level. Four partial algebras have been introduced for the formal definition of these levels (see figure 4.8).

See figure 4.11 for an overview of the concepts defined and their relations. The operations of the four algebras operate on the population of the depicted meta schema.

Remark 4.5.1

Note that the $AMRH$ object type is a schema type. In PSM schema types are defined as object types, one instance of which will be a population of the underlying schema. In this case, one instance of the object type $AMRH$ is a population of the object types enclosed by the $AMRH$ object type. \square

We did not discuss the notion of forgetting in this chapter, as this would introduce a whole range of new problems. Some of these problems are identified in subsection 9.2.1.1.

Chapter 5

Evolving Object Role Models

The history of every major Galactic Civilisation tends to pass through three distinct and recognizable phases, those of Survival, Inquiry and Sophistication, otherwise known as the How, Why and Where phases. For instance, the first phase is characterised by the question How can we eat? the second by the question Why do we eat? and the third by the question Where shall we have lunch?

*From: "The Hitch Hiker's Guide to the Galaxy",
Douglas Adams, Pan Books Ltd.*

5.1 Applying the General Theory

In this chapter we present EVORM (EVolutionary Object Role Modelling), a data modelling technique for evolving application domains. EVORM provides a more concrete way of modelling for evolving application domains, and is the result of applying the general evolution theory to the object role modelling technique PSM ([HW93]). This chapter is based mainly on [PW94].

PSM is a conceptual data modelling technique, introduced as a common denominator ([BBMP95]) for object-role models, such as ER ([Che76]), Extended ER versions (e.g. [EGH⁺92]), NIAM ([NH89]), INFOMOD ([JG87]), FDM ([Shi81]), IFO ([AH87]) and PM ([BHW91]). Additionally, PSM offers some powerful concepts that are particularly suited to modelling data with a complex structure, such as hypermedia documents ([Wig90]), meta models ([Wel88], [KW92], [Ver93a]), and CAD (Computer Aided Design) systems. Thus, application of the general theory to PSM provides a good test case for this general theory.

In section 3.1, we stated that when applying the general evolution theory to a concrete (data) modelling technique, the modelling technique must provide a typing system that conforms to the typing axioms of the general theory, and to well-formedness rules for versions of models using the *IsPop* and *IsSch* predicates. The axioms defined by the general theory consisted of three classes: ISU, AMU and AMV. The relationship between these axiom classes is depicted in figure 3.1 (page 38). The well-formedness of application model versions is captured by the *IsAM* predicate, and the well-formedness of histories is captured by *IsAMH*. EVORM requires the introduction of four new classes of axioms:

- EU: typing mechanism and information structure universe
- TR: type relatedness
- EV: version well-formedness
- EEW: evolution well-formedness
- P: population well-formedness

The EU and TR axioms together provide a typing mechanism, defining an information structure universe that conforms to the ISU axioms. The IsPop, IsSch and IsEvol predicates, are defined by the P, EV and EEW axioms respectively. In figure 5.1, these new classes are related to the original framework of the general theory depicted in figure 3.1 (page 38). In a proper application model history (IsAMH) all axioms will hold.

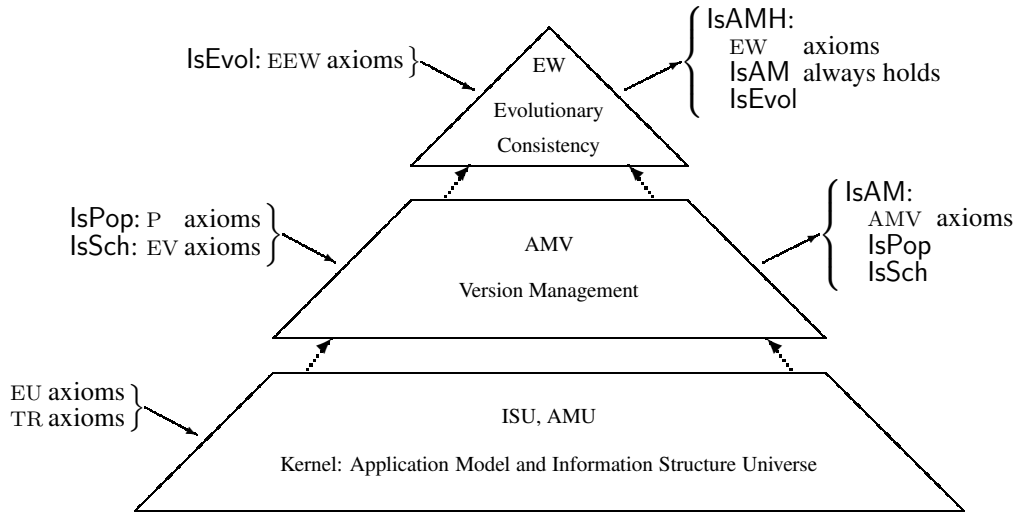


Figure 5.1: Axiomatic framework, revisited

The structure of this chapter is as follows. An informal discussion of the PSM/EVORM modelling concepts is given in the next section. The information structure universe for EVORM is introduced in section 5.3, leading to the EU and TR axioms. The EVORM application model universe and its evolution, defined by the P, EV and EEW axioms, is defined in section 5.4.

5.2 Informal Introduction to PSM

An informal introduction to the Predicate Set Modelling Technique (PSM) is provided in this section. The motivation of PSM from a theoretical point of view, its formal foundations and its expressiveness can be found in [HW93], the motivation from a practical point of view is presented in [HPW92b].

Although we have already employed PSM to depict meta schemas, we offer the reader a short overview of the modelling concepts in PSM before providing the formal definition of EVORM. The informal introduction is based on [HW93], [HPW92b] and [HPW93].

5.2.1 Basic Data Modelling Concepts

One of the key concepts in data modelling is the concept of relationship type or fact type. In both ER related and NIAM related data modelling techniques, a relation type is considered to be an association between object types. The graphical representation of a binary relation type R between object types X_1 and X_2 in the NIAM style is shown in figure 5.2.

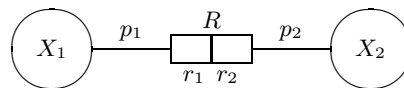


Figure 5.2: A NIAM relation type

The basic building element of fact types is the connection between an object type and a role, called a *predicator*. In figure 5.2, p_1 is the predicator connecting X_1 to r_1 , and p_2 the predicator that connects X_2 to r_2 . In PSM, a fact type is considered to be a set of predicators. As a consequence, a relation type is an association between predicators, rather than between objects types. Fact types are also regarded as object types. If a fact type plays a role, the fact type is deemed *objectified*.

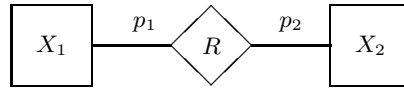


Figure 5.3: The corresponding ER diagram

Two special kinds of object types are entity types and label types. The distinguishing property is that labels can, contrary to entities, be represented directly on a communication medium. As a result, label types are also called *concrete object types*. All other object types are called *abstract*, and their instances are not directly representable. As usual, a clear distinction is made between concrete object types and abstract object types. The gap between these concrete and abstract object types can only be crossed by special binary fact types. These fact types correspond to *bridge types* in NIAM and *attribute types* in ER. It remains to say that each entity type must be identifiable in terms of label types.

5.2.2 Specialisation

Specialisation, referred to as subtyping in NIAM, is a mechanism for representing one or more (possibly overlapping) subtypes of an object type. Specialisation is applied only when certain facts are to be recorded for specific instances of an object type. For example, suppose that you are interested in the cars adults, i.e. persons with an age greater than or equal to 18, own. This situation is captured by the PSM schema in figure 5.4. In this figure the black dot on the object type Person is an example of a total role constraint, it expresses that each instance of Person should play the role has. The arrow above this role is an example of a uniqueness constraint and expresses the rule that instances of Person play the role has at most once. The formal semantics of these graphical constraint types can be found in [BHW91] and [Hof93]. An overview of the drawing conventions is included in appendix A.

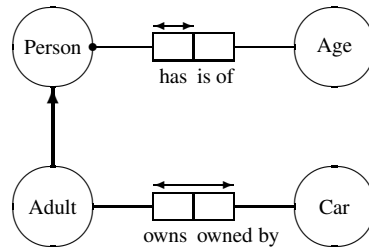


Figure 5.4: Example of specialisation

A specialisation relation between a subtype and a supertype implies that the instances of the subtype are also instances of the supertype (each Adult is also a Person). It is required for proper specialisation, that subtypes be defined in terms of one or more of their supertypes. Such a decision criterion is referred to as a *subtype defining rule* ([BHW91]), for figure 5.4 the subtype defining rule for Adult is expressed as (in LISA-D):

$$\text{Adult} = \text{Person has Age} \geq 18$$

Identification of subtypes is derived from their supertypes. Therefore if, in the ongoing example, Persons are identified by a name, then Adults are also identified by that name.

Specialisation relations are organised in specialisation hierarchies. A specialisation hierarchy is in fact not a hierarchy in the strict sense, but an acyclic directed graph with a unique top. This top is referred to as the *pater familias* (see [DMV88]). In the example of figure 5.4, the pater familias of Adult is Person.

Object types inherit all properties from their ancestors in the specialisation hierarchy. This characteristic of specialisation excludes non-entity types (e.g. fact types) occurring as subtypes. Consider for example that a ternary fact type is a subtype of a binary fact type. Clearly this leads to a contradiction. No problems occur when non-entity types themselves are specialised. Consequently, non-entity types always act as pater familias, for an in depth discussion of specialisation we refer to [HHO92].

5.2.3 Generalisation

Generalisation is a mechanism that allows for the creation of a new object type as a generic type for other object types. The constituent object types in a generalisation are called the *specifiers* of the generalised object type. As a result, the *generalised object type* is covered by its constituent object types. This means that every instance of any specifier is also an instance of the generalised object type. Another consequence is that the identification of a generalised object type is determined by the identification of its specifiers. Contrary to what its name suggests, generalisation is *not* the inverse of specialisation. Specialisation and generalisation originate from different axioms in set theory ([HW93]) and therefore have a different expressive power.

Typically for generalisation it is required that the generalised object type is covered by its constituent object types (or specifiers). Therefore, a decision criterion as used in specialisation (the subtype defining rule) is not necessary. Further, properties are inherited ‘upward’ in a generalisation hierarchy instead of ‘downward’, which is the case for specialisation (see also [AH87]). This also implies that the identification of a generalised object type depends on the identification of its specifiers. From the nature of generalisation it is apparent that a non-entity type cannot be a generalised object type.

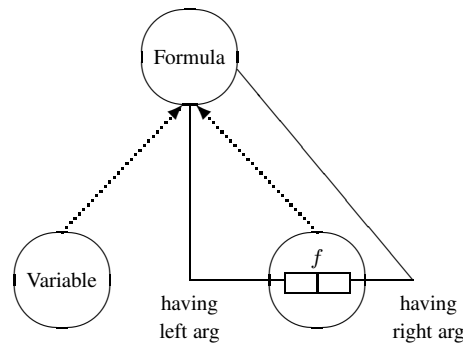


Figure 5.5: Formulas, an example of generalisation

An example of generalisation is provided in figure 5.5. A formula may be either a single variable, or constructed by some function (say f) from simpler formulas. It is clear that instances from the object type Formula inherit the structure (identification) from the specifier from which they originate (Variable or f).

This example also shows that generalisation can be used to define recursive object types. This is not possible in the IFO data model ([AH87]), where object types are hierarchical structures.

5.2.4 Power types

The concept of *power type* in PSM forms the data modelling pendant of power sets in conventional set theory ([Lev79]). This notion corresponds to the notion of grouping introduced in the IFO data model. An instance of a

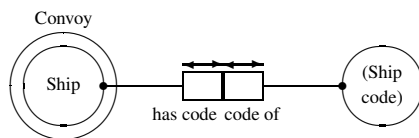
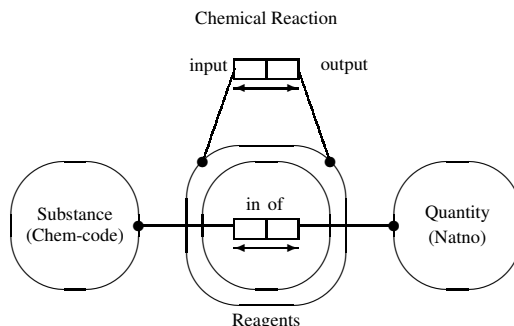


Figure 5.6: A simple example of a power type



input	output												
<table> <tr> <td>over</td><td>in</td></tr> <tr> <td>H₂</td><td>2</td></tr> <tr> <td>O₂</td><td>1</td></tr> </table>	over	in	H ₂	2	O ₂	1	<table> <tr> <td>over</td><td>in</td></tr> <tr> <td>H₂O</td><td>2</td></tr> </table>	over	in	H ₂ O	2		
over	in												
H ₂	2												
O ₂	1												
over	in												
H ₂ O	2												
<table> <tr> <td>over</td><td>in</td></tr> <tr> <td>CH₃ - CH₂OH</td><td>2</td></tr> <tr> <td>O₂</td><td>1</td></tr> </table>	over	in	CH ₃ - CH ₂ OH	2	O ₂	1	<table> <tr> <td>over</td><td>in</td></tr> <tr> <td>CH₃ - CHO</td><td>2</td></tr> <tr> <td>H₂O</td><td>2</td></tr> </table>	over	in	CH ₃ - CHO	2	H ₂ O	2
over	in												
CH ₃ - CH ₂ OH	2												
O ₂	1												
over	in												
CH ₃ - CHO	2												
H ₂ O	2												

Figure 5.7: Chemical reactions

power type is a set of instances of its *element type*. Such an instance is identified by its elements, just as a set is identified by its elements in set theory (axiom of extensionality).

The convoy problem (based on [HM81]), depicted in figure 5.6 is an example of power typing. Here, the object type Convoy is a power type with as an element type Ship. As a result, each instance of object type Convoy is a set of instances of Ship. Convoys are identified by their constituent ships, whereas ships are identified by a Ship-code, which is a label type. To distinguish label types from entity types in diagrams, label type names are parenthesised. The convoy problem is *not* expressible in terms of a NIAM or ER schema (see [HPW92b]), without violating the conceptualisation principle.

A further illustrative example of the elegance of power types, is provided in figure 5.7. The modelled universe of discourse deals with simple chemical reactions. A chemical reaction takes a set of input substances with their associated quantities, and produces a set of output substances in corresponding quantities.

5.2.5 Sequence types

Sequence typing offers the opportunity to represent sequences, built from an underlying element type. This notion is not elementary in PSM, as it is expressible in terms of generalisation (see [HW93]). Nonetheless, the concept of sequence type is treated as an independent concept in the formalisation of EVORM, as this has proven to be more elegant.

Sequence types can be seen as power types with an associated order. Instances of sequence types are tuples of arbitrary length of instances from the underlying element type. As an example, consider a freight train as depicted in

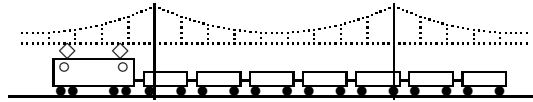


Figure 5.8: An example Freight Train

figure 5.8. A train is identified by a train code, and consists of a locomotive followed by a sequence of freight cars. This universe of discourse is modelled in the information structure diagram of figure 5.9.

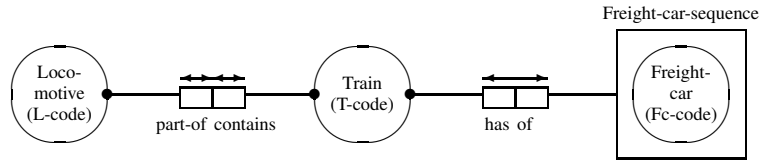


Figure 5.9: The train composition administration

5.2.6 Schema types

The need for decomposition in large systems has been generally recognised. A well-known example is the decomposition mechanism for activity graphs ([Sch84]). In an activity graph, both processes and data may be subject to decomposition. Data modelling techniques usually do not provide such a decomposition mechanism. *Schema objectification* is introduced in PSM for this purpose. Schema objectification, resulting in schema types, is a construction mechanism allowing for the decomposition of large schemata into, objectified, subschemata. Instances of such object types are then populations of their corresponding schemas.

Though in [HW93] it has been argued that schema typing is not an elementary concept, it is considered to be an independent concept here for the same reason given for sequence types in the previous section.

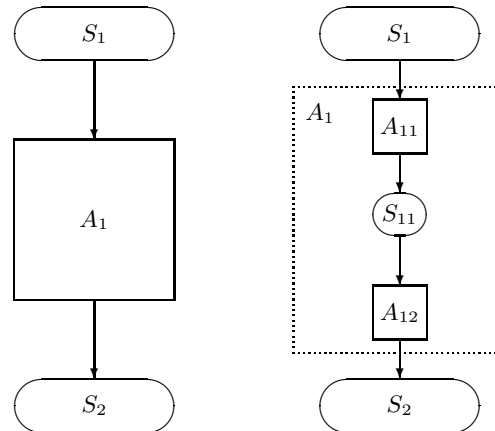


Figure 5.10: Sample activity graphs

As an example of schema typing, we consider activity graphs. Activity graphs are bipartite directed graphs consisting of activities and states. The direction of the arrow between an activity and a state indicates whether that state is an input or output of that activity. Activities and states can be decomposed into other activity graphs. In figure 5.10 the rightmost activity graph shows the decomposition of activity A_1 . The meta model of activity graphs is depicted in

figure 5.11. As can be seen, an activity graph is an objectified schema consisting of activities, states and input and output relations. The binary relations between activity and activity graph and state and activity graph represent the decomposition relation.

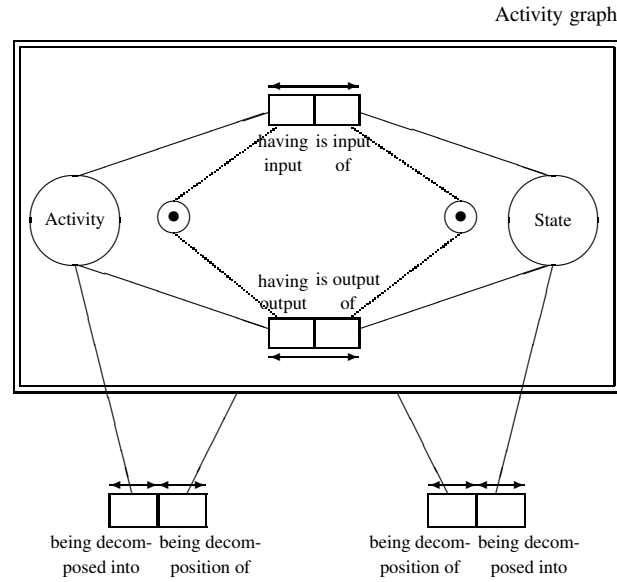


Figure 5.11: Meta-model of activity graphs

5.3 EVORM Information Structure Universe

We introduce the information structure universe for the modelling technique EVORM in this section. Since EVORM is based on PSM, most of the PSM axioms are also applicable to EVORM since they define what constitutes a proper EVORM schema version. In this chapter we will not discuss all axioms for EVORM schemata. For a detailed discussion of these axioms, see [Hof93].

5.3.1 Information structure universe

The following components can be identified in the EVORM information structure universe:

1. A finite set \mathcal{P} of *predicators*.
2. A nonempty set \mathcal{O} of *object types*.
3. A set of *label types* \mathcal{L} . As every label type is an object type we have: $\mathcal{L} \subseteq \mathcal{O}$.
4. A set \mathcal{E} of *entity types*. Any entity type is also an object type, so: $\mathcal{E} \subseteq \mathcal{O}$.
5. A partition \mathcal{F} of the set \mathcal{P} . The elements of \mathcal{F} are called *fact types*. All fact types are object types, so: $\mathcal{F} \subseteq \mathcal{O}$.
6. A set \mathcal{G} of *power types*. Every power type is an object type, hence: $\mathcal{G} \subseteq \mathcal{O}$.
7. A set \mathcal{S} of *sequence types*. Each sequence type is an object type, therefore: $\mathcal{S} \subseteq \mathcal{O}$.
8. A set \mathcal{C} of *schema types*. Any schema type is also an object type, so: $\mathcal{C} \subseteq \mathcal{O}$.

9. A function $\text{Base} : \mathcal{P} \rightarrow \mathcal{O}$. The base of a predicate is the object type part of that predicate.
10. A function $\text{Elt} : \mathcal{G} \cup \mathcal{S} \rightarrow \mathcal{O}$. This function yields the *element type* of a power type or sequence type.
11. A partial order $\text{IdfBy} \subseteq \mathcal{O} \times \mathcal{O}$ on object types, capturing the inheritance hierarchy.
12. A partial order $\text{Spec} \subseteq \text{IdfBy}$ specifying that part of the identification hierarchy concerned with specialisation.
13. A partial order $\text{Gen} \subseteq \text{IdfBy}$ specifying that part of the identification hierarchy concerned with generalisation.
14. A relation $\prec \subseteq \mathcal{C} \times \mathcal{O}$, describing the decomposition of schema types.
15. Not every set of object types leads to a correct schema. Therefore, the relation (set) $\text{IsSch} \subseteq \mathcal{O}$ designates which schemas are correct. This predicate is defined formally, together with the IsPop predicate, in the next section providing the well-formedness rules for EVORM (see figure 5.1 (page 80)).

For predicates we define the auxiliary function $\text{Fact} : \mathcal{P} \rightarrow \mathcal{F}$, yielding the fact type in which a given predicate is contained. This function is defined by: $\text{Fact}(p) = f \iff p \in f$.

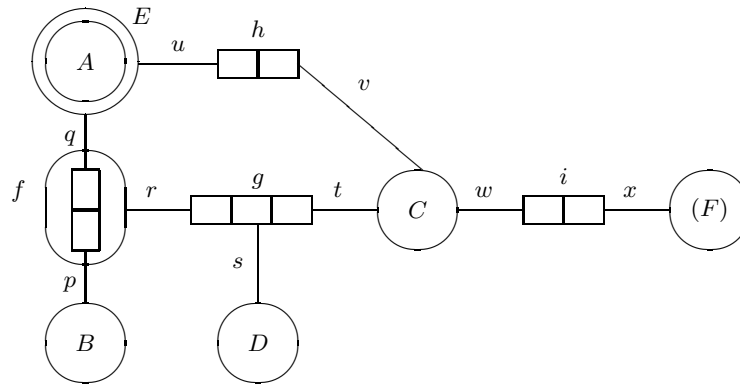


Figure 5.12: Example information structure

Example 5.3.1

Figure 5.12 shows an information structure diagram visualising the information structure that consists of:

$$\begin{array}{ll}
 \mathcal{P} &= \{p, q, r, s, t, u, v, w, x\} & \mathcal{S} &= \emptyset \\
 \mathcal{O} &= \{A, B, C, D, E, F, f, g, h, i\} & \mathcal{C} &= \emptyset \\
 \mathcal{F} &= \{f, g, h, i\} & \mathcal{E} &= \{A, B, C, D\} \\
 \mathcal{G} &= \{E\} & \mathcal{L} &= \{F\}
 \end{array}$$

where $f = \{p, q\}$, $g = \{r, s, t\}$, $h = \{u, v\}$, $i = \{w, x\}$. With respect to the predicates: $\text{Base}(p) = B$, $\text{Base}(q) = A$, $\text{Base}(r) = f$, etc. Finally, $\text{Elt}(E) = A$, Spec and Gen are empty. \square

A meta model for EVORM is depicted in figure 5.13. This meta model does not contain any evolution related concepts, and should be regarded as a subschema of the meta model of the general theory provided in figure 3.10 providing a refinement of the concepts depicted.

We briefly formulate rules for this information structure universe in terms of EU axioms in the remainder of this section. For a more complete discussion of these axioms, see [HW93] and [Hof93]. Further, the IsSch predicate is defined in terms of the EU-axioms.

Label types, entity types, fact types, power types, sequence types and schema types are all interpreted differently:

[EU1] $\mathcal{L}, \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{S}, \mathcal{C}$ form a partition of \mathcal{O} .

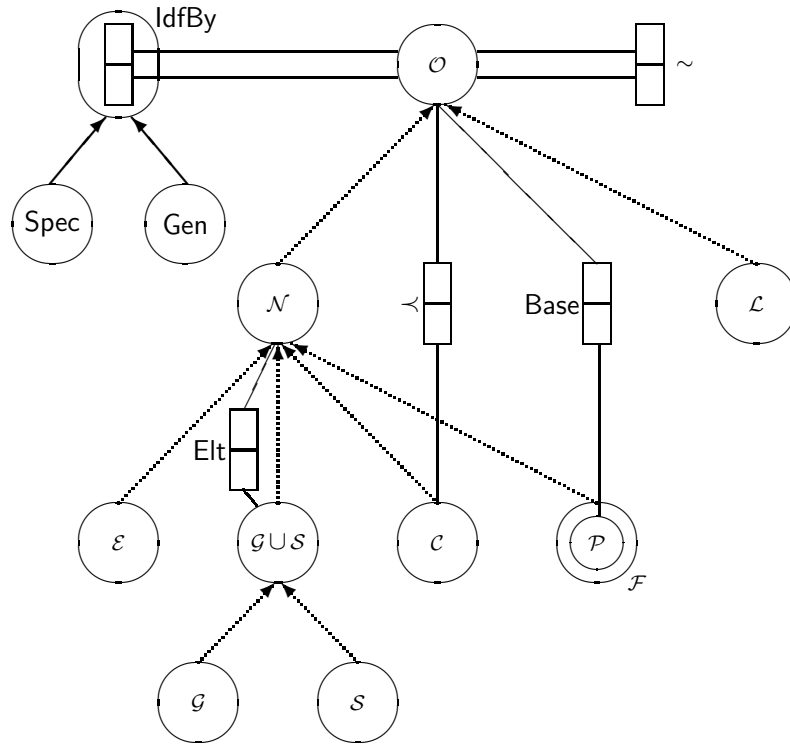


Figure 5.13: Meta model of EVORM

5.3.2 Abstract and concrete object types

Bridge types establish the connection between abstract and concrete object types. The term $\text{Bridge}(f)$ qualifies fact type f as a *bridge type*, and is an abbreviation of the expression:

$$\exists_{p,q} [f = \{p, q\} \wedge \text{Base}(p) \in \mathcal{L} \wedge \text{Base}(q) \notin \mathcal{L}]$$

The set of all bridge types in the information structure universe is denoted by \mathcal{B} . The strict separation between the concrete and abstract level is expressed by the following rules. Label types may only participate in bridge types:

$$[\text{EU2}] \quad \text{Base}(p) \in \mathcal{L} \Rightarrow \text{Bridge}(\text{Fact}(p))$$

The predicates that constitute a bridge type $b = \{p, q\}$ can be extracted from the bridge type, by the operators concr and abstr . These operators are defined by:

$$\begin{aligned} \text{concr}(b) &\triangleq p \text{ such that: } p \in b \wedge \text{Base}(p) \in \mathcal{L} \\ \text{abstr}(b) &\triangleq q \text{ such that: } q \in b \wedge \text{Base}(q) \notin \mathcal{L} \end{aligned}$$

Example 5.3.2

In figure 5.12, i is a bridge type with $\text{concr}(i) = x$ and $\text{abstr}(i) = w$. □

5.3.3 Power types

The element type of a power type is found by the function Elt . The relation between a power type x and its element type $\text{Elt}(x)$ is recorded in the fact type $\in_{x, \text{Elt}(x)}$. In general an implicit fact type $\in_{x,y}$ is used as a bridge between

complex object types x (power types, sequence types and schema types), and their elementary types y . This fact type is defined by:

$$\in_{x,y} \triangleq \{ \in_{x,y}^c, \in_{x,y}^e \}$$

where $\text{Base}(\in_{x,y}^c) = x$ and $\text{Base}(\in_{x,y}^e) = y$. With respect to power types, this relation is assumed to be available for each power type.

Usually, implicit fact types are not drawn in an information structure diagram, only if such a fact type is subject to constraints, or used in an objectification, does it need to be made explicit. Note that, in this way, power types corresponds to a polymorphic type constructor, and the fact type $\in_{x,\text{Elt}(x)}$ to an associated polymorphic access operator. The strict separation between abstract and concrete object types prohibits label types occurring as an element type:

[EU3] $\text{Elt}(x) \notin \mathcal{L}$

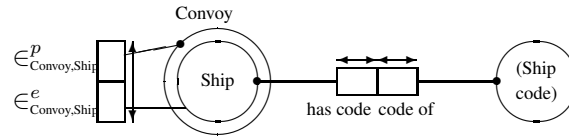


Figure 5.14: Example power type

Example 5.3.3

Figure 5.14 extends figure 5.6 with the implicit fact type associated with power types. The implicit fact types $(\in_{\text{Convoy},\text{Ship}})$ are usually not displayed. \square

5.3.4 Sequence types

The element type of a sequence type is also found by the function Elt . The relation between a sequence type x and its element type $\text{Elt}(x)$ is recorded by the implicit fact type $\in_{x,\text{Elt}(x)}$. Contrary to power types, this relation $\in_{x,\text{Elt}(x)}$ is augmented with the position of the element in the sequence, via the implicit fact type $@_{x,\text{Elt}(x)} \triangleq \{ @_{x,\text{Elt}(x)}^s, @_{x,\text{Elt}(x)}^i \}$, where $\text{Base}(@_{x,\text{Elt}(x)}^s) = \in_{x,\text{Elt}(x)}$ and $\text{Base}(@_{x,\text{Elt}(x)}^i) = \text{I}$. The object type I is the domain for indexes in sequence types. Usually the natural numbers are used for this purpose. The index type is assumed to be a label type ($\text{I} \in \mathcal{L}$), which is assumed to be totally ordered and to have a least element. Note that axiom EU3 also applies to sequence types.

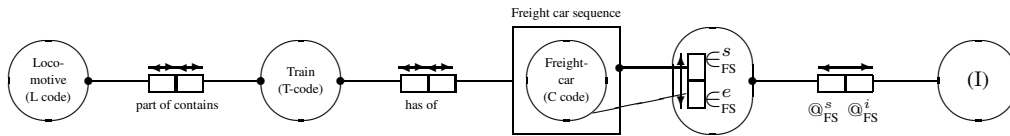


Figure 5.15: The train composition administration

Example 5.3.4

Figure 5.15 extends figure 5.9 with the implicit fact types associated with sequence typing. In this figure FS is a shorthand for Freight-car-sequence, Freight-car. \square

5.3.5 Schema types

Schema types can be decomposed into an underlying information structure via the relation \prec , with the convention that $x \prec y$ is interpreted as x is decomposed into y or y is part of the decomposition of x .

This underlying information structure \mathcal{I}^x for a schema type x is derived from the object types into which x is decomposed: $\mathcal{O}^x \triangleq \{y \in \mathcal{O} \mid x \prec y\}$. Analogously the special object classes \mathcal{F}^x , \mathcal{G}^x , \mathcal{S}^x , \mathcal{C}^x and \mathcal{E}^x can be derived.

The implicit fact type $\in_{x,y}$ is associated with each schema type x and each object type y in its decomposition. These fact types enable the transition from a composed object to an object from its decomposition.

5.3.6 Identification hierarchy

The *identification hierarchy* in EVORM is defined as a partial order (asymmetric and transitive) ldfBy on object types, with the convention that $a \text{ ldfBy } b$ is interpreted as: *a inherits its identification from b*. Note that an identification hierarchy only deals with inheritance of identification via specialisation or generalisation. As non-root object types inherit the structure of their parents, they are atomic, i.e. entity or label types. The abstract and concrete object types should not be mixed up:

$$[\text{EU4}] \text{ (strictness) } \text{ldfBy} \subseteq \mathcal{E} \times (\mathcal{O} \setminus \mathcal{L}) \cup \mathcal{L} \times \mathcal{L}$$

The nature of a partial order is expressed by:

$$[\text{EU5}] \text{ (asymmetry) } x \text{ ldfBy } y \Rightarrow \neg y \text{ ldfBy } x$$

$$[\text{EU6}] \text{ (transitivity) } x \text{ ldfBy } y \text{ ldfBy } z \Rightarrow x \text{ ldfBy } z$$

We define ldfBy_1 as the one step counterpart of ldfBy :

$$x \text{ ldfBy}_1 y \triangleq x \text{ ldfBy } y \wedge \neg \exists z [x \text{ ldfBy } z \text{ ldfBy } y]$$

In EVORM, all object types in the identification hierarchy have direct ancestors:

$$[\text{EU7}] \text{ (direct ancestors) } x \text{ ldfBy } y \Rightarrow x \text{ ldfBy}_1 y \vee \exists p [x \text{ ldfBy}_1 p \text{ ldfBy } y]$$

The finite depth of the identification hierarchy in EVORM is expressed by the following schema of induction:

[EU8] (identification induction) If F is a property for object types, such that:

$$\forall y \in \mathcal{O} \left[\forall x: y \text{ ldfBy}_1 x [F(x)] \Rightarrow F(y) \right]$$

then $\forall x \in \mathcal{O} [F(x)]$

The identification hierarchy is a result of specialisation and generalisation:

[EU9] (complete span)

$$\begin{aligned} x \text{ ldfBy}_1 y &\Rightarrow x \text{ Gen } y \vee x \text{ Spec } y \\ x \text{ ldfBy } y &\Leftarrow x \text{ Gen } y \vee x \text{ Spec } y \end{aligned}$$

In the next subsections, the relations Spec and Gen will be refined in such a way that Spec and Gen turn out to be filter relations of ldfBy .

5.3.7 Specialisation

The concept of specialisation is modelled as a partial order (asymmetric and transitive) Spec on object types. The intuition behind $a \text{ Spec } b$ is: a is a specialisation of b , or a is a subtype of b .

[EU10] (*transitivity completeness*) If $x \text{ ldfBy } y \text{ ldfBy } z$ then:

$$x \text{ Spec } y \text{ Spec } z \iff x \text{ Spec } z$$

Note that the asymmetry of Spec follows from the asymmetry of ldfBy , as $\text{Spec} \subseteq \text{ldfBy}$. On specialisation hierarchies, we define the *pater familias* relation. This relation represents the root relation, if we restrict ourselves to specialisation based inheritance. The pater familias relation is identified by: $\sqcap(x, y) \triangleq (x \text{ Spec } y \vee x = y) \wedge \neg \text{spec}(y)$ where $\text{spec}(x)$ is a shorthand for $\exists_y [x \text{ Spec } y]$. Each specialisation hierarchy, contrary to generalisation, has a unique top element. This is stipulated by the following axiom:

[EU11] (*unique pater familias*) $\sqcap(x, y) \wedge \sqcap(x, z) \Rightarrow y = z$

This axiom allows us to regard the pater familias relation \sqcap as a partial function, and to write $\sqcap(x) = y$ instead of $\sqcap(x, y)$. In a later subsection we provide a proof for the existence of a pater familias for all object types, thus proving that \sqcap is a total function on \mathcal{O} .

5.3.8 Generalisation

The concept of generalisation is introduced as a partial order Gen . The expression $a \text{ Gen } b$ stands for: a is a generalisation of b , or b is a specifier of a . In the sequel $\text{gen}(x)$ will be used as an abbreviation for $\exists_y [x \text{ Gen } y]$.

[EU12] (*transitivity completeness*) If $x \text{ ldfBy } y \text{ ldfBy } z$ then:

$$x \text{ Gen } y \text{ Gen } z \iff x \text{ Gen } z$$

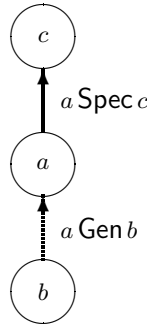


Figure 5.16: Conflicting generalisation and specialisation

Generalisation and specialisation can be conflicting due to their inheritance structure. To avoid such conflicts, generalised object types are required to be pater familias:

[EU13] $\text{gen}(x) \Rightarrow \neg \text{spec}(x)$

This axiom is illustrated in figure 5.16. Object type a inherits its identification from c and b , which leads to a contradiction.

Basic specifiers of a generalised object type are defined analogously to the pater familias of a specialised object type: $\sqcup(x, y) \triangleq (x \text{ Gen } y \vee x = y) \wedge \neg \text{gen}(y)$. Note that uniqueness of basic specifiers is not required. As a shorthand, we will write $\sqcup(x)$ for the set $\{y \mid \sqcup(x, y)\}$.

5.3.9 Type relatedness

Intuitively, object types can, for several reasons, have values in common in some instantiation. For example, each value of object type x will, in any instantiation, also be a value of object type $\sqcap(x)$. As another example, suppose $x \text{ Gen } y$, then any value of y in any population will also be a value of x . A third example, where object types may share values is when two power types have element types that may share values.

Formally, for EVORM, type relatedness is captured by a binary relation \sim on \mathcal{O} . Two object types are type related if, and only if, this can be proven from the following derivation rules:

$$[\text{TR1}] \quad x \in \mathcal{O} \vdash x \sim x$$

$$[\text{TR2}] \quad x \sim y \vdash y \sim x$$

$$[\text{TR3}] \quad y \text{ ldfBy } x \wedge x \sim z \vdash y \sim z$$

$$[\text{TR4}] \quad x, y \in \mathcal{G} \wedge \text{Elt}(x) \sim \text{Elt}(y) \vdash x \sim y$$

$$[\text{TR5}] \quad x, y \in \mathcal{S} \wedge \text{Elt}(x) \sim \text{Elt}(y) \vdash x \sim y$$

$$[\text{TR6}] \quad \mathcal{O}_x = \mathcal{O}_y \vdash x \sim y$$

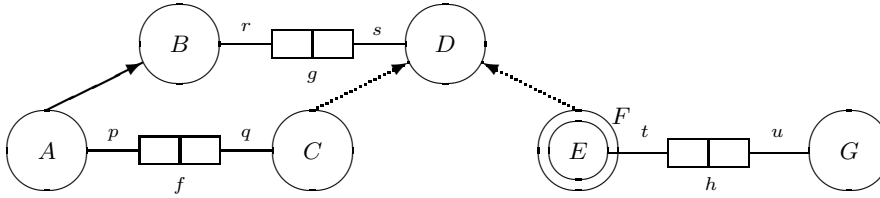


Figure 5.17: Example information structure

Example 5.3.5

In figure 5.17 the only object types that are type related are A and B , C and D , and E and G . □

5.3.10 Valid EVORM versions

In this subsection, we discuss what constitutes a *proper information structure version* in EVORM. All axioms on information structure versions defined by the general theory also apply to information structure versions of EVORM. Therefore, we restrict ourselves here to EVORM specific rules.

For EVORM, two classes of well-formedness rules on information structure versions are defined. Prior to doing so, however, a definition must be given of an information structure version in terms of the object types alive at a given point in time. Let \mathcal{O}_t be a set of object types spanning an information structure version at point in time t , derived from an application model history H . From now on, we assume that \mathcal{O}_t is (implicitly) extended with all implicit object types. The associated EVORM information structure is then:

$$\mathcal{I}_t \triangleq \langle \mathcal{F}_t, \mathcal{G}_t, \mathcal{S}_t, \mathcal{C}_t, \mathcal{E}_t, \mathcal{L}_t, \mathcal{P}_t \rangle$$

The set of fact types (including the implicit fact types) in the EVORM information structure version is defined as: $\mathcal{F}_t \triangleq \mathcal{F} \cap \mathcal{O}_t$. The other components are derived analogously. The set of predicates, on the other hand, is defined as: $\mathcal{P}_t \triangleq \bigcup \mathcal{F}_t$.

The first class of well-formedness rules is engaged with the completeness of the version, with respect to object types with an underlying decomposition. This underlying decomposition must, at least partially, be present in the version under consideration. For the information structure derived from \mathcal{O}_t , we therefore have the following *time-conformity rules*:

$$[\text{EV1}] \quad p \in \mathcal{P}_t \Rightarrow \text{Base}(p) \in \mathcal{O}_t$$

$$[\text{EV2}] \quad x \in \mathcal{G}_t \cup \mathcal{S}_t \Rightarrow \text{Elt}(x) \in \mathcal{O}_t$$

$$[\text{EV3}] \quad x \in \mathcal{C}_t \Rightarrow \exists y \in \mathcal{O}_t [x \prec y]$$

Note that for a schema type it is, so far, only required that some underlying object type must be present. The second class of well-formedness rules therefore poses more strict rules on the versions of the decomposition of a schema type. For schema types, each underlying information structure version should be a proper information structure version on its own:

$$[\text{EV4}] \quad x \in \mathcal{C}_t \Rightarrow \text{IsSch}(\mathcal{O}_t^x)$$

where $\mathcal{O}_t^x = \{y \in \mathcal{O}_t \mid x \prec y\}$. The IsSch predicate is defined in definition 5.3.1.

Further, the foundation of live axiom (axiom AMV5 (page 50)) should also hold for decomposition:

$$[\text{EV5}] \quad \text{If } x \in \mathcal{C}_t, \text{ then the property } DL_t^x \text{ defined as } DL_t^x(y) \triangleq y \in \mathcal{O}_t^x \text{ is reflected by } \text{ldfBy}.$$

The well-formedness axioms on information structure versions allow us to define what we regard as a good EVORM information structure version:

Definition 5.3.1

$$\text{IsSch}(\mathcal{O}_t) \triangleq \mathcal{O}_t \text{ adheres to the EV axioms}$$

□

This definition provides the well-formedness predicate for (schema) versions in EVORM (see figure 3.1). In the next section, we will prove that EVORM provides a proper typing mechanism.

One might argue that IsSch is an intuitively wrong name for the above predicate since the predicate states a property of an information structure, and *not* of the conceptual schema consisting of the constraints *and* the information structure. However, the IsSch predicate can easily be made more specific by taking rules regarding identification and subtyping into consideration. Rules which can only be defined in the context of a conceptual schema as a whole, as they depend on the constraints present in the conceptual schema. The importance of identifiability of object types, and the intricate problems of cyclic dependencies in subtype defining rules, have been discussed in [HW93], [HPW93], and [Hof93] for PSM schemas. As each EVORM information structure together with its associated constraints is a PSM schema, the rules from PSM apply directly.

5.4 EVORM Application Model Universe

In this section, EVORM is described as an application model universe. We show how EVORM spans an information structure universe, and prove that this universe is a proper information structure universe defined in section 2.2. The information structure universe of EVORM is more detailed than that of the general theory, as more concepts are recognised. As a result, versions of the EVORM information structure universe have associated (along with the AMV axioms) more rules regarding well-formedness. In EVORM two additional classes of rules can be identified.

The first class, discussed in the previous section, takes well-formedness of the information structure into account, leading to the predicate IsSch . The second class poses restrictions on populations, resulting in the predicate IsPop , and will be discussed at the end of this section. Note that the predicates IsSch , IsPop and the AMV axioms form the definition of the predicate IsAM (see section 3.4). With respect to well-formedness of evolution (see section 3.5), no extra rules besides the EW axioms are assumed.

The application model universe for EVORM is defined by the tuple (see definition 3.3.1 (page 43)):

$$\mathcal{U}_{\Sigma}^{\text{orm}} \triangleq \langle \mathcal{U}_{\text{IS}}^{\text{orm}}, \mathcal{D}, \Omega, \text{IsPop}, \gamma, \text{IsConstrSet}, \mu, \text{IsActMod}, \llbracket \rrbracket, \text{IsEvol}, \text{Depends} \rangle$$

In the next subsection we describe the components $\mathcal{U}_{\text{IS}}^{\text{orm}}$, \mathcal{D} , Ω and IsPop . As we restrict ourselves in this chapter to the modelling of information structures, the components γ , IsConstrSet , μ , IsActMod , $\llbracket \rrbracket$ and Depends fall outside the scope of this chapter. We also describe extra rules for populations, leading to the definition of the underlying domain Ω for instances.

5.4.1 The Information Structure Universe

The elements for the EVORM information structure universe have been introduced in subsection 5.3.1. In this subsection we show how this universe fits within the general theory of section 2.2. The EVORM information structure universe is spanned by:

$$\mathcal{U}_{\text{IS}}^{\text{orm}} \triangleq \langle \mathcal{L}, \mathcal{N}, \sim, \rightsquigarrow, \text{IsSch} \rangle$$

\mathcal{L} is the set of label types that built the information structure universe (see section 2.2). The set \mathcal{N} of *non-lexical object types* consists of:

$$\mathcal{N} \triangleq \mathcal{E} \cup \mathcal{F} \cup \mathcal{G} \cup \mathcal{S} \cup \mathcal{C}$$

The type relatedness relation \sim for EVORM has already been defined by the TR axioms. The inheritance hierarchy \rightsquigarrow of EVORM, corresponds to the relation ldfBy^{-1} . The relation IsSch has been introduced in definition 5.3.1.

5.4.1.1 Verifying the axioms

The EVORM information structure is a proper information structure that conforms to the general evolution theory, and thus provides a correct typing system (see figure 3.1):

Theorem 5.4.1 $\mathcal{U}_{\text{IS}}^{\text{orm}}$ is an information structure universe.

Proof:

All axioms ISU1 to ISU8 hold:

1. The axioms ISU1 and ISU2 (page 23) follow directly from axioms TR1 and TR2 (page 91).
2. From axiom EU4 (page 89) and the TR axioms, axiom ISU3 (page 23) directly follows.
3. Axioms ISU4 to ISU7 correspond to axioms EU5 to EU8.
4. Axiom ISU8 is treated in the two lemmas given below.

□

To prove the correctness of the \sim and \rightsquigarrow relation of EVORM with respect to the axioms of the general evolution theory, all that remains to be done is to prove that axiom ISU8 also holds for the EVORM \sim and \rightsquigarrow relation. This is proven in the following two lemmas:

Lemma 5.4.1 EVORM type relatedness is preserved by \rightsquigarrow :

$$x \sim y \wedge y \rightsquigarrow z \Rightarrow x \sim z$$

Proof:

As $\rightsquigarrow \triangleq \text{ldfBy}^{\leftarrow}$, axiom TR3 (page 91) yields directly the result. \square

Lemma 5.4.2 EVORM type relatedness is reflected by \rightsquigarrow :

$$x \sim y \wedge \neg \text{lsRoot}(y) \Rightarrow \exists_z [x \sim z \wedge z \rightsquigarrow y]$$

Proof:

We apply parent induction to x .

Induction hypothesis:

$$x \sim y \wedge \neg \text{RootOf}(y) \Rightarrow \exists_z [x \sim z \wedge z \rightsquigarrow y]$$

In the induction step, we distinguish three cases:

1. If $x = y$, then:

$$\neg \text{lsRoot}(y) \equiv \{\text{definition of RootOf}\}$$

$$\exists_z [z \rightsquigarrow y] \Rightarrow \{x = y\}$$

$$\exists_z [z \rightsquigarrow x \wedge z \rightsquigarrow y] \Rightarrow \{\text{corollary 2.2.1 (page 26)}\}$$

$$\exists_z [x \sim z \wedge z \rightsquigarrow y]$$

2. If $x \neq y$, the validity of $x \sim y$ is a result of the application of the derivation rules for \sim (see section 5.3.9). Consider a minimal length proof of $x \sim y$. As y is not a root object type, we can conclude $y \in \mathcal{E}$. As a result, the last step in the proof of $x \sim y$ is either an application of axiom TR2 or axiom TR3.

This leads to two more cases:

(a) If axiom TR3 has been applied last, then:

$$\exists_p [\text{ParentOf}(p, x) \wedge p \sim y] \Rightarrow \{\text{induction hypothesis}\}$$

$$\exists_{p,z} [\text{ParentOf}(p, x) \wedge p \sim z \wedge z \rightsquigarrow y] \Rightarrow \{\text{axiom TR3}\}$$

$$\exists_z [x \sim z \wedge z \rightsquigarrow y]$$

(b) If axiom TR2 has been applied last, then $y \sim x$ has been proven by axiom TR3.

As a result $\exists_p [\text{ParentOf}(p, y) \wedge p \sim x]$, which results directly in: $\exists_p [x \sim p \wedge p \rightsquigarrow y]$

\square

As stated before, we are able to prove that Spec^{-1} and Gen^{-1} are filter relations for ldfBy^{-1} . Even more, they are an identification signature for \rightsquigarrow :

Theorem 5.4.2 The relations Spec^{-1} and Gen^{-1} are an identification signature of ldfBy^{-1} .

Proof:

To prove this we have to show the validity of the F axioms (page 32) demonstrating that Spec^{-1} and Gen^{-1} are filter relations of ldfBy^{-1} . Further, we have to prove that Spec^{-1} and Gen^{-1} form an identification signature (definition 2.4.1 (page 35)). For this purpose, three items have to be proven:

1. Axiom F1 follows for both relations, directly from axiom EU10 and EU12.
2. Axiom F2 can be proven for Gen as follows:

Let $x \text{ ldfBy}_1 p$ and $x \text{ ldfBy}_1 q$, and $x \text{ Gen } p$.

From axiom EU9 (page 89) follows $x \text{ Gen } q \vee x \text{ Spec } q$.

Applying axiom EU13 (page 90), and $x \text{ Gen } p$, yields $x \text{ Gen } q$.

The proof for Spec goes analogously.

3. From axiom EU9, and axiom EU13, follows directly:

$$x \text{ ldfBy}_1 y \Rightarrow \exists!_{R \in \{\text{Gen}, \text{Spec}\}} [R(x, y)]$$

□

5.4.1.2 Results

As a result of these last two theorems, the properties of the identification hierarchy as proven in section 2.3 also hold for the identification hierarchies for EVORM models, i.e. they are now properties ‘for free’.

A first result are the following two theorems, which are an application of lemma 2.3.1 to the definition of \sqcap (see subsection 5.3.7) and \sqcup (see subsection 5.3.8):

Theorem 5.4.3

$$\sqcap(x) = \sqcap(y) \Rightarrow x \sim y$$

Proof:

Applying lemma 2.3.1 (page 29) for \cong_{Spec} yields:

$$v \text{ RootOf}_{\text{Spec}} x \wedge v \text{ RootOf}_{\text{Spec}} y \Rightarrow x \sim y$$

Combining the definitions of $\text{RootOf}_{\text{Spec}}$ and \sqcap , together with the uniqueness of a pater familias, leads to:

$$\sqcap(x) = \sqcap(y) \Rightarrow x \sim y$$

□

Theorem 5.4.4

$$\sqcup(x) \cap \sqcup(y) \neq \emptyset \Rightarrow x \sim y$$

Proof:

Applying lemma 2.3.1 (page 29) for \cong_{Gen} yields:

$$v \text{ RootOf}_{\text{Gen}} x \wedge v \text{ RootOf}_{\text{Gen}} y \Rightarrow x \sim y$$

Combining the definitions of $\text{RootOf}_{\text{Gen}}$ and \sqcup leads to:

$$\sqcup(x) \cap \sqcup(y) \neq \emptyset \Rightarrow x \sim y$$

□

A further result is that \sqcap and \sqcup are total functions. This follows by applying corollary 3.4.2 (page 50) for $\text{RootOf}_{\text{Spec}}$ and $\text{RootOf}_{\text{Gen}}$ respectively. Finally, theorem 5.4.3 can even be strengthened to:

Theorem 5.4.5

$$\sqcap(x) = \sqcap(y) \wedge y \sim z \Rightarrow x \sim z$$

Proof:

Apply theorem 2.3.5 (page 30) with \cong_{Spec}

□

When applying corollary 3.4.2 (page 50) to Gen and Spec respectively, we have for information structure versions, the following corollaries:

Corollary 5.4.1 $a \in \mathcal{O}_t \Rightarrow \sqcup(a) \cap \mathcal{O}_t \neq \emptyset$

Corollary 5.4.2 $a \in \mathcal{O}_t \Rightarrow \sqcap(a) \in \mathcal{O}_t$

5.4.2 Populations of Information Structure Versions

A version of a population at t , derived from an application model history, is a mapping:

$$\text{Pop}_t : \mathcal{O} \rightarrow \wp(\Omega)$$

where Ω is the universe of instances that can occur in the population of the information structure universe. For EVORM, the set of instances Ω is defined in terms of two base sets.

5.4.2.1 Universe of instances

The first set provides the concrete instances. An information structure can only be populated if a link is established between label types and concrete domains. The instances of label types then come from their associated concrete domain. This link has been formally established by the function $\text{Dom}_t : \mathcal{L}_t \rightarrow \mathcal{D}$. The range of this function, i.e. \mathcal{D} , is the set of concrete domains (e.g. string, natno). The sets in \mathcal{D} form the carriers of a many sorted algebra $\langle \mathcal{D}, F \rangle$, where F is the set of operations (e.g. +) on the sorts in \mathcal{D} .

The second set provides the atomic abstract instances: Θ . They are used to populate the root entity types. The *universe of instances* Ω is inductively defined as the smallest set satisfying:

1. $\bigcup \mathcal{D} \subseteq \Omega$. Instances from the sorts in the many sorted algebra are elements of the universe of instances.
2. $\Theta \subseteq \Omega$, where Θ is an abstract (countable) domain of (unstructured) values that may occur in the population of entity types.
3. If $x_1, \dots, x_n \in \Omega$ and $p_1, \dots, p_n \in \mathcal{P}$ then also $\{p_1 : x_1, \dots, p_n : x_n\} \in \Omega$. The set $\{p_1 : x_1, \dots, p_n : x_n\}$ denotes a mapping, assigning x_i to each predictor p_i . These mappings are intended for the population of fact types.
4. If $x_1, \dots, x_n \in \Omega$ then also $\{x_1, \dots, x_n\} \in \Omega$. Sets of instances may occur as instances of power types.
5. If $x_1, \dots, x_n \in \Omega$ then $\langle x_1, \dots, x_n \rangle \in \Omega$. Sequences of instances are used as instances of sequence types. The i -th element of a sequence $\langle x_1, \dots, x_n \rangle$, i.e. x_i , can be derived using projection, denoted as: $\langle x_1, \dots, x_n \rangle_{<i>}$.
6. If $X_1, \dots, X_n \subseteq \Omega$ and $O_1, \dots, O_n \in \mathcal{O}$ then also $\{O_1 : X_1, \dots, O_n : X_n\} \in \Omega$. Assignments of sets of instances to object types are also valid instances, and they are intended for the populations of schema types.

5.4.2.2 Implicit object types

The Pop_t function, derived from an application model history H , does not yet provide a population of the implicit object types. These populations are derived from the other object types by the following derivation rules:

[IO1] (*power base rule*) If $x \in \mathcal{G}$ and $y = \text{Elt}(x)$, then:

$$\text{Pop}_t(\in_{x,y}) = \{ \{ \in_{x,y}^c : u, \in_{x,y}^e : v \} \mid u \in \text{Pop}_t(x) \wedge v \in u \}$$

[IO2] (*sequence decomposition rule*) If $x \in \mathcal{S}$ and $y = \text{Elt}(x)$, then:

$$\text{Pop}_t(\in_{x,y}) = \{ \{ \in_{x,y}^c : u, \in_{x,y}^e : v \} \mid u \in \text{Pop}_t(x) \wedge \exists i \in \mathbb{N} [u_{<i>} = v] \}$$

[IO3] (*sequence indexing rule*) If $x \in \mathcal{S}$ and $y = \text{Elt}(x)$, then:

$$\text{Pop}_t(@_{x,\text{Elt}(x)}) = \{ \{ @_{x,y}^s : u, @_{x,y}^i : v \} \mid u \in \text{Pop}_t(\in_{x,y}) \wedge u(\in_{x,y}^c)_{<v>} = u(\in_{x,y}^e) \}$$

[IO4] (*indexing rule*) If $x \in \mathcal{S}$ and $y = \text{Elt}(x)$, then:

$$\text{Pop}_t(\text{I}) = \{ u(@_{x,y}^i) \mid u \in \text{Pop}_t(@_{x,\text{Elt}(x)}) \}$$

[IO5] (*decompositor rule*) If $x \prec y$, then:

$$\text{Pop}_t(\in_{x,y}) = \{ \{ \in_{x,y}^c : u, \in_{x,y}^e : v \} \mid u \in \text{Pop}_t(x) \wedge v \in u(y) \}$$

Henceforth, we assume Pop_t to be extended with the populations of derived object types.

5.4.2.3 Well-formed populations

The population of a root entity type is a set of values, taken from the abstract domain Θ .

[P1] If $x \in \mathcal{E}_t$ and $\text{lsRoot}(x)$ then: $\text{Pop}_t(x) \subseteq \Theta$.

The population of a fact type is a set of tuples. A tuple y in the population of a fact type x is a mapping of all its predicates to values of the appropriate type. This is referred to as the *conformity rule*:

[P2] If $x \in \mathcal{F}_t$ and $y \in \text{Pop}_t(x)$ then: $y : x \rightarrow \Omega \wedge \forall_{p \in x} [y(p) \in \text{Pop}_t \cdot \text{Base}(p)]$.

The population of a power type consists of (nonempty) sets of instances of the corresponding element type. This is called the *power type rule*:

[P3] If $x \in \mathcal{G}_t$ and $y \in \text{Pop}_t(x)$ then: $y \in \mathcal{O}^+ \cdot \text{Pop}_t \cdot \text{Elt}(x)$.

The population of a sequence type consists of (nonempty) sequences of instances of the corresponding element type. This is called the *sequence type rule*:

[P4] If $x \in \mathcal{S}_t$ and $y \in \text{Pop}_t(x)$ then: $y \in (\text{Pop}_t \cdot \text{Elt}(x))^+$.

The population of a composition type consists of populations of the underlying information structure. This is called the *decomposition rule*:

[P5] If $x \in \mathcal{C}_t$ and $y \in \text{Pop}_t(x)$ then: $\text{lsPop}(\mathcal{I}_t^x, y)$.

The nature of generalisation requires the following rule, stating that the population of a specifier is a subset of the population of the generalised object type:

[P6] $x \text{ Gen } y \Rightarrow \text{Pop}_t(y) \subseteq \text{Pop}_t(x)$

Below, we show that the population of a generalised object type is exactly the union of the populations of its specifiers.

The P axioms lead to the definition of the lsPop predicate, completing the well-formedness rules on versions of EVORM models:

Definition 5.4.1

$\text{lsPop}(\mathcal{O}_t, \text{Pop}_t) \triangleq \text{Pop}_t$ and \mathcal{O}_t adhere to the P axioms

□

Example 5.4.1

A sample population version of the information structure version of figure 5.12 is:

$$\begin{aligned}
 \text{Pop}(A) &= \{a_1, a_2\} & \text{Pop}(f) &= \{\{p : b_1, q : a_1\}, \{p : b_1, q : a_2\}\} \\
 \text{Pop}(B) &= \{b_1\} & \text{Pop}(g) &= \{\{r : \{p : b_1, q : a_1\}, s : d_1, t : c_1\}\} \\
 \text{Pop}(C) &= \{c_1\} & \text{Pop}(h) &= \{\{u : \{a_1\}, v : c_1\}, \{u : \{a_1, a_2\}, v : c_1\}\} \\
 \text{Pop}(D) &= \{d_1\} & \text{Pop}(i) &= \{\{w : c_1, x : 17\}\} \\
 \text{Pop}(E) &= \{\{a_1\}, \{a_1, a_2\}\} \\
 \text{Pop}(F) &= \{17\}
 \end{aligned}$$

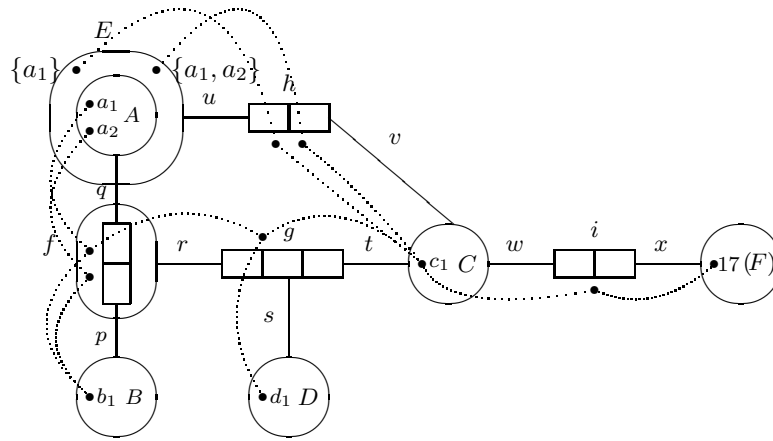


Figure 5.18: Graphic representation of a population

In this example, it is assumed that the concrete domain of label type F is the set of natural numbers. In the above population 17 comes from this domain and is the only label instance. The instances a_1 , a_2 , b_1 , c_1 and d_1 come from the abstract domain Θ and are considered to be non-denotable by a user. Note that if the instance $\{w : c_2, x : 17\}$ is added to the population of fact type i the conformity rule is violated, since c_2 is not an element of $\text{Pop}(C)$. This population is graphically represented in figure 5.18. The population of the implicit fact type $\in_{E,A}$ can be derived to be:

$$\text{Pop}_t(\in_{E,A}) = \left\{ \begin{array}{l} \{\in_{E,A}^c : \{a_1\}, \in_{E,A}^e : a_1\}, \\ \{\in_{E,A}^c : \{a_1, a_2\}, \in_{E,A}^e : a_1\}, \\ \{\in_{E,A}^c : \{a_1, a_2\}, \in_{E,A}^e : a_2\} \end{array} \right\}$$

□

5.4.2.4 Results

Now let Σ_t be a correct application model version ($\text{IsAM}(\Sigma_t)$). For such a Σ_t , the AMV and P axioms hold, as well as the theorems proven for the general theory. We focus on the population in such a version, and reconsider the results from subsection 3.4.4.

Respecting the specialisation hierarchy is reflected by the *specialisation rule*, which follows directly from lemma 3.4.2 (page 52) and axiom EU11 (page 90):

Corollary 5.4.3 $\text{Pop}_t(x) \subseteq \text{Pop}_t(\sqcap(x))$

This rule does not require that instances of subtypes have to fulfill the subtype defining rule associated to the involved subtype. A subtype defining rule is defined as an information descriptor (see [HPW93]).

Respecting the generalisation hierarchy is reflected by the *generalisation rule*, which follows from lemma 3.4.2 (page 52) and axiom P6 (page 97):

Corollary 5.4.4 $\text{Pop}_t(x) = \bigcup_{y \in \sqcup(x)} \text{Pop}_t(y)$

The *generalisation rule*, which is clearly a derivation rule, requires that the population of a generalised object type (x) is completely covered by the populations of its specifiers.

5.4.3 Well-formedness of evolution

The IsEvol predicate is used to enforce extra, EVORM dependent, well-formedness rules on application model histories. Since EVORM is a general data modelling technique, such extra rules are a matter of taste. One sensible well-formedness rule would be:

[EEW1] (*monotonous composition*) If $h_1, h_2 \in H_{type}$ and $h_1, h_2 \downarrow t, \triangleright t$, then:

$$h_1(t) \text{ PartOf } h_2(t) \Rightarrow h_1(\triangleright t) \text{ PartOf } h_2(\triangleright t)$$

where:

$$x \text{ PartOf } y \triangleq x = \text{Elt}(y) \vee y \prec x \vee \exists_p [\text{Base}(p) = x \wedge \text{Fact}(p) = y]$$

This rule states that the ‘construction order’ of object types may not be reversed in one evolution step. It is interesting to note the similarity between the above axiom and axiom EW3 (page 57).

Remark 5.4.1

When replacing the \leadsto of the general evolution theory with a general notion of ‘decomposition’ of object types, axiom EEW1 and axiom EW3 could be merged into one single axiom.

A promising approach in this respect can be found in [FHL97], where a category theoretical ([BW90b]) description of the concepts underlying PSM/EVORM is provided. \square

The IsEvol predicate for an application model history H can simply be defined by:

$$\text{IsEvol}(H) \triangleq H \text{ adheres to the EEW axioms}$$

Although we provided only one EEW axiom, one could add more axioms as desired.

5.5 Conclusions

In this chapter we applied the general evolution theory to the concrete data modelling technique PSM to obtain EVORM. We introduced four classes of axioms for EVORM:

- EU: typing mechanism and information structure universe
- TR: type relatedness
- EV: version well-formedness
- EEW: evolution well-formedness
- P: population well-formedness

The relation between these axiom classes, and the original axioms from the evolution theory is depicted in figure 5.1.

In retrospect, four phases can be identified in the application of the general evolution theory to PSM. These steps can generally be used when applying the general evolution theory to concrete modelling techniques. The steps are:

1. *Identify the information structure and/or application model universe.*

The information structure universe for EVORM has been defined in section 5.3 as well as subsection 5.4.1, the application model universe is given in the introduction of section 5.4.

2. *Prove the correctness of the identified universe with respect to the general theory.*

The correctness of the EVORM information structure universe has been proven in subsection 5.4.1.1. As EVORM does not add any other new aspects to the application model universe than the EVORM information structure universe, no further prove of its correctness needed to be provided.

3. *Are extra, technique dependent, well-formedness rules needed?*

A set of EVORM specific axioms is introduced in subsection 5.3.10, that defines what constitutes a correct version of an EVORM model. Further, subsection 5.4.2 deals with extra rules for versions of populations of EVORM information structures, as contained in application model versions. Finally, a well-formedness axiom for evolution of EVORM models is provided in subsection 5.4.3.

4. *Interpret and apply the “theorems for free”.*

These results can be found for EVORM in subsection 5.4.1.2, and 5.4.2.4.

The next step is to find a suitable verbalisation mechanism for the concepts of the theory, i.e., a proper way of communicating. A way of communicating is presented in the next chapter, that leads to the formulation of queries and updates in a semi-natural language. This language will be strongly related to the language used by domain experts to describe the underlying universe of discourse.

Chapter 6

Information Disclosure in an Evolving Environment

One of the major problems encountered in time travel is not that of accidentally becoming your own father or mother. There is no problem involved in becoming your own father or mother that a broad minded and well adjusted family can't cope with. There is also no problem about changing the course of history – the course of history does not change because it all fits together like a jigsaw. All the important changes have happened before the things they were supposed to change and it all sorts itself out in the end.

The major problem is quite simply one of grammar, and the main work to consult in this matter is Dr Dan Streetment's Time Traveller's Handbook of 1001 Tense Formations. It will tell you for instance how to describe something that was about to happen to you in the past before you avoided it by time-jumping forward two days in order to avoid it.

*From: "The Restaurant at the End of the Universe",
Douglas Adams, Pan Books Ltd.*

6.1 Introduction

In this chapter, which is based on [HPW93] and [PW95b], we introduce a manipulation language for evolving information systems. The language introduced is related to the natural language approach (way of thinking) underlying NIAM (Natural language based Information Analysis Method). This method investigates the grammar of communication in a universe of discourse. Usually this grammar is depicted as an information structure diagram (NIAM or ER schema).

This chapter describes the language Elisa-D, which is based on this grammar. Queries in this language have a direct meaning in the universe of discourse, while natural language expressions are easily formalised. A most important aspect of information systems, is their ability to support information disclosure. In subsection 1.3.1, we have already stated the following requirement for the disclosure of information in (evolving) information systems:

*The system should provide an adequate disclosure mechanism for the retrieval of **all** stored information.*

Stored information refers to the application model as a whole (see figure 1.7 (page 9)). For evolving information systems, the history of all stored information (the application model) is also subject of discourse. The latter poses

extra requirements on the disclosure mechanism. As the underlying (information) structure of the stored information changes in the course of time, traditional query languages such as SQL are inadequate.

In some approaches to evolving information systems, a manipulation language for relational models is extended with historical operations, both for populations, and structural aspects of an application model. An example of this approach can be found in [MS90a], in which an algebra is presented allowing relational tables to evolve by changing their arity. This direction is similar to the ORION project ([BK87], [KBC⁺89]), in that a manipulation language is extended with operations supporting schema evolution. Nevertheless, it is not clear whether these languages provide an adequate disclosure mechanism for the information stored in all versions of the underlying information structure, together with the populations conforming to these information structures. We consider the following (not necessarily orthogonal) criteria to be relevant for disclosure (query) languages:

- [L1] (*expressiveness*) The expressiveness of a language provides an indication of the semantical richness of that language.
- [L2] (*suitability*) Suitability addresses the match between the concepts underlying the disclosure language, and the concepts used in the universe of discourse.
- [L3] (*elegance*) The elegance of a language is concerned with the ease with which searchers can formulate their information need.
- [L4] (*supportiveness*) What (automated) support can be provided to the searcher in formulating information needs?

Expressiveness will only be addressed briefly. The elegance and suitability of a language can only be validated empirically, and will therefore not be considered here. The supportiveness issue has been studied before in [BPW93], where a two-level hypermedia approach, an interactive process of query formulation called *query by navigation*, is described. We will elaborate more on this approach, in the context of evolving information systems, in chapter 8.

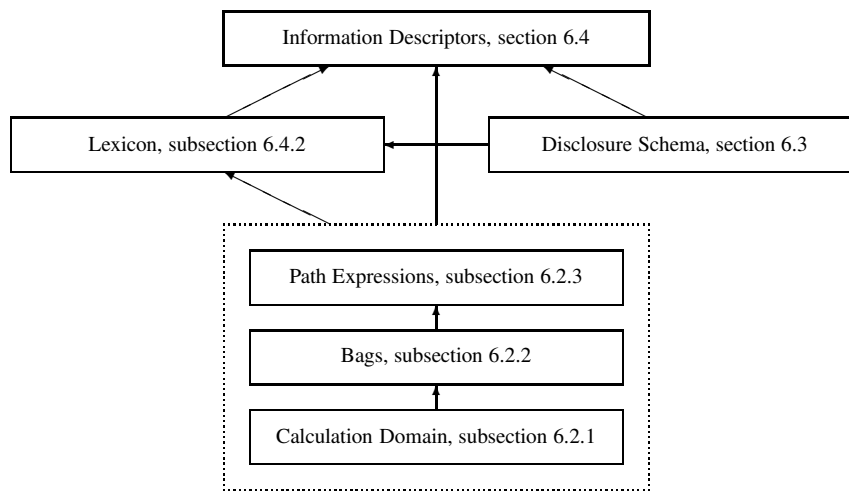


Figure 6.1: The structure of this chapter

The structure of this chapter is best explained in terms of figure 6.1. We start out by defining a semantic domain for Elisa-D. The core of this semantic domain is formed by a calculation domain, providing a calculation mechanism determining the quantities in which elements occur as the result of Elisa-D queries. Using this calculation domain, a generalised notion of bags (or multisets) is defined.

These bags are, in their turn, applied to the definition of path expressions. Path expressions correspond to paths through the information structure of the application under consideration. This information structure, corresponds to the disclosure schema, which provides the structure of *all* stored information. The semantics of Elisa-D queries (*information descriptors*), is defined in terms of these path expressions. Information descriptors, which also correspond to paths through the information structure, resemble natural language more closely than path expressions do, and therefore have a direct intuitive meaning.

To translate information descriptors to path expressions, a lexicon is needed providing verbalisations of the concepts in the disclosure schema. References to the corresponding sections have been provided in figure 6.1.

6.2 Semantic Domain

The semantics of Elisa-D is defined in terms of an underlying semantic domain. This section is concerned with this underlying domain. The main syntactic category in Elisa-D is formed by the information descriptors. As stated before, information descriptors correspond to paths through the information structure, chaining object types to each other. The semantics of these information descriptors is defined in terms of operations on binary relations.

The semantics of LISA-D ([HPW93]), the non-evolutionary predecessor of Elisa-D, has been expressed by bags. Bags differ from sets from ordinary set theory in that an element may occur more than once, albeit only a natural number of times (see e.g. [Lew85], [Lev79], or [Par90]). In the definition of the semantics of Elisa-D we will try to be more general, by presuming the existence of an underlying *calculation domain*.

The choice of the calculation domain determines whether the result of a query will, for instance, be a set, a bag, or a set of elements with associated probabilities. With this calculation domain, we therefore define a generalised class of *bags* allowing for elements with occurrence quantities as defined by the calculation domain. The generalised form of bags, is then employed to define the *path expressions*, which will eventually be used to define the information descriptors of Elisa-D.

6.2.1 Calculation Domain

There exist a number of ways to calculate the frequency of elements (tuples) in the result of a query. In relational algebra, which is based on normal sets, an element in the result cannot occur more than once. SQL is based on bags, which means that the elements in the result can occur a natural number of times. The quantity of elements in the outcome of a query is (in general) not even restricted to natural numbers. In an expert system, one may associate some kind of measure providing the confidence (belief/disbelief) of the system with respect to the result. Further, in an information retrieval system one may want to associate relevance, with respect to the information request submitted to an information retrieval system, to the stored objects.

In this subsection, we define the notion of *calculation domain*. When defining the semantics of Elisa-D in terms of this calculation domain, Elisa-D may easily be adapted to other requirements by defining a calculation domain suited for new requirements. A calculation domain is defined by the following algebra:

$$\mathbb{C} \triangleq \langle \mathbb{Q}, \cap, \cup, -, \times, 0_{\mathbb{Q}}, 1_{\mathbb{Q}}, \text{ds}, < \rangle$$

The intuition between the components of this algebra is as follows:

1. The set \mathbb{Q} is the domain for the quantities in which an instance occurs in a population or query result.
2. The operations $\cap, \cup, -, \times : \mathbb{Q} \times \mathbb{Q} \rightarrow \mathbb{Q}$, each model one kind of operation on quantities. Each of these operations is used to define the generalised bag operations in the next subsection. The intuition behind the operations of the algebra is as follows:
 - (a) \cap is the intersection of two quantities.
 - (b) \cup is the union of two quantities.

- (c) $-$ is the difference of two quantities.
- (d) \times is the multiplication of two quantities.
- 3. The constants: $0_{\mathbb{Q}}$ and $1_{\mathbb{Q}}$ denote the null value and the neutral value for the \cap operation of this algebra respectively.
- 4. The $ds : \mathbb{Q} \rightarrow \{0_{\mathbb{Q}}, 1_{\mathbb{Q}}\}$ operator is used to neutralise the quantity.
- 5. The relation $<$ defines a complete partial order on \mathbb{Q} . With this relation, we can define \leq in the obvious way.

The exact laws, such as commutativity and associativity, to which these operations adhere, depend on the chosen quantification domains. Some examples of such algebras are:

1. We can reason about elements of normal sets by defining $\mathbb{Q} \triangleq \mathbb{B}$ and:

$$\begin{array}{llll}
 x \cap y & \triangleq & x \wedge y & 1_{\mathbb{Q}} \triangleq true \\
 x \cup y & \triangleq & x \vee y & 0_{\mathbb{Q}} \triangleq false \\
 x - y & \triangleq & x \wedge \neg y & ds(x) \triangleq x \\
 x \times y & \triangleq & x \wedge y & x < y \triangleq x \wedge \neg y
 \end{array}$$

This definition clearly makes \cap, \cup and \times associative and commutative operations.

2. The following definition results in a computational domain for traditional bags, with $\mathbb{Q} \triangleq \mathbb{N}$:

$$\begin{array}{llll}
 x \cap y & \triangleq & \min(x, y) & 1_{\mathbb{Q}} \triangleq 1 \\
 x \cup y & \triangleq & x + y & 0_{\mathbb{Q}} \triangleq 0 \\
 x - y & \triangleq & \max(x - y, 0) & ds(x) \triangleq \text{if } x > 0 \text{ then } 1 \text{ else } 0 \text{ fi} \\
 x \times y & \triangleq & x y & x < y \triangleq x < y
 \end{array}$$

3. Probabilities are obtained by defining $\mathbb{Q} \triangleq [0, 1] \times [0, 1]$, where $x = \langle x_b, x_d \rangle$ is to be interpreted as the *belief* and *disbelief* in the presence of the element under consideration.

$$\begin{array}{llll}
 \langle x_b, x_d \rangle \cap \langle y_b, y_d \rangle & \triangleq & \langle \min \{x_b, y_b\}, \max \{x_d, y_d\} \rangle \\
 \langle x_b, x_d \rangle \cup \langle y_b, y_d \rangle & \triangleq & \langle \max \{x_b, y_b\}, \min \{x_d, y_d\} \rangle \\
 \langle x_b, x_d \rangle - \langle y_b, y_d \rangle & \triangleq & \langle x_b y_d, x_d y_b \rangle \\
 \langle x_b, x_d \rangle \times \langle y_b, y_d \rangle & \triangleq & \langle x_b y_b, x_d y_d \rangle \\
 1_{\mathbb{Q}} & \triangleq & \langle 1, 0 \rangle \\
 0_{\mathbb{Q}} & \triangleq & \langle 0, 1 \rangle \\
 ds(\langle x_b, x_d \rangle) & \triangleq & \text{if } x_b x_d < 0.5 \text{ then } 0_{\mathbb{Q}} \text{ else } 1_{\mathbb{Q}} \text{ fi} \\
 \langle x_b, x_d \rangle < \langle y_b, y_d \rangle & \triangleq & x_b x_d < y_b y_d
 \end{array}$$

The above definitions, for \cap, \cup and \times correspond to the definitions of probability combinations in inference networks discussed in [LG91], and expert systems such as MYCIN ([BS84], [HRWL83]).

One could also introduce combinations of the above calculation domains, for instance frequency and probability of elements. In the definition of Elisa-D we will presume the usage of traditional bags, and the associated properties on the operations.

An omission in the above discussed approach, needing further research, is the richness of the set of operations in the algebra. For instance, in the expert system field, in particular belief nets, an additional operation exists: co-conclusion, combining the probabilities of two (independent) proofs of the truth of a proposition (see also [LG91]). Further, general properties for calculation domains may be stated and proven.

6.2.2 Generalised Bags

Generalised bags over an underlying domain X are elegantly introduced as functions: $X \rightarrow \mathbb{Q}$, assigning to each $x \in X$ its quantity. In the remainder, whenever we refer to bags, we refer to this generalised notion of bags. In the definitions of operations on bags, the λ -calculus notation provided by [Bar84], is employed. For instance $\lambda x.x^2$ is the polynomial function assigning x^2 to each x -value. With a \mathbb{C} algebra, and a given set X , the following domain can be associated:

$$\mathbb{C}(X) \triangleq X \rightarrow \mathbb{Q}$$

On this domain, a set of operations is defined, which together provide us with a generalised notion of bags. We start with a comprehension schema for bags (see [Boi92]). Let C be a partial function $C : X \rightarrow \mathbb{Q}$, then:

$$\{[e \uparrow^q \mid C(e, q)]\} \triangleq \lambda_{e \in X} \text{.if } \exists_q [C(e, q)] \text{ then that } q \text{ else } 0_{\mathbb{Q}} \text{ fi}$$

Bag comprehension can be used for intentional denotations of bags. Extensional denotations are defined by:

$$\begin{aligned} \{[a]\} &\triangleq \{[a \uparrow^1 \mathbb{Q}]\} \\ \{[a_1, \dots, a_n]\} &\triangleq \{[a_1]\} \cup \dots \cup \{[a_n]\} \end{aligned}$$

As a shorthand, $e \in^q M$ is used rather than $M(e) = q$, and $e \in M$ rather than $M(e) > 0$.

The following comparison operators for bags can be defined:

$$\begin{aligned} N \subseteq M &\iff \forall_x [N(x) \leq M(x)] \\ N \subset M &\iff N \subseteq M \wedge N \neq M \end{aligned}$$

This allows for the definition of the power set of a multiset:

$$\wp(X) \triangleq \{[Y \uparrow^1 \mathbb{Q} \mid Y \subseteq X]\}$$

If θ is a n -ary operation on \mathbb{Q} , then this operation can be generalised to $\mathbb{C}(X)$ by:

$$\theta(N_1, \dots, N_n) \triangleq \lambda_{e \in X} \theta(N_1(e), \dots, N_n(e))$$

By applying this to the $\cap, \cup, -, \times$, ds operations, these operations are generalised to bags. It is not hard to prove that if $\theta_1, \theta_2 \in \{\cap, \cup, -, \times\}$ are associative, distributive or commutative on \mathbb{Q} , then they are so on $\mathbb{C}(X)$.

Further, we define: $0_X \triangleq \lambda_{x \in X} 0_{\mathbb{Q}}$, and $1_X \triangleq \lambda_{x \in X} 1_{\mathbb{Q}}$, where 0_X corresponds to the empty set for bags. Coercions from bags to sets and vice versa are defined by the following functions:

$$\begin{aligned} \text{Set}(N) &\triangleq \{n \mid n \in N\} \\ \text{Multi}(S) &\triangleq \{[n \uparrow^1 \mathbb{Q} \mid n \in S]\} \end{aligned}$$

The ‘number’ of elements in a bag is counted by:

$$|N| \triangleq \text{sum}_{x \in X} N(x)$$

In this thesis, a useful class of multisets operations operates on bags over binary tuples, so: $\mathbb{C}(X \times X)$. We start by defining the following coercion operations between $\mathbb{C}(X)$ and $\mathbb{C}(X \times X)$:

$$\begin{aligned} \text{Sqr}(N) &\triangleq \{[\langle x, x \rangle \uparrow^q \mid x \in^q N]\} \\ \pi_1(N) &\triangleq \lambda_{x \in X} \bigcup_{y \in X} N(x, y) \\ \pi_2(N) &\triangleq \lambda_{y \in X} \bigcup_{x \in X} N(x, y) \end{aligned}$$

We define three operations for $\mathbb{C}(X \times X)$:

$$\begin{aligned} N^{\leftarrow} &\triangleq \lambda_{\langle x, y \rangle \in X \times X}. N(y, x) \\ N \circ M &\triangleq \lambda_{\langle x, y \rangle \in X \times X}. \bigcup_{a \in X} N(x, a) \times M(a, y) \\ N \diamond M &\triangleq \lambda_{\langle x, y \rangle \in X \times X}. \bigcup_{a, b \in X} N(x, a) \times M(y, b) \end{aligned}$$

where N^{\leftarrow} corresponds to the reverse relation, $N \circ M$ to the concatenation of N and M , and $N \diamond M$ to the head-head combinations of N and M . We also define the following operation, being the bag pendant of a union of a set of sets:

$$\biguplus N \triangleq \lambda_{x \in X}. \bigcup_{A \in N, x \in A} N(A)$$

Note that N is a bag of sets.

By making assumptions on the underlying domains X some more useful operations can be introduced. If a complete partial order $<$ exists on X , then the following two operations can be defined:

$$\begin{aligned} \max(N) &\triangleq x \text{ such that: } x \in N \wedge \forall_{y \in N} [y \leq x] \\ \min(N) &\triangleq x \text{ such that: } x \in N \wedge \forall_{y \in N} [x \leq y] \end{aligned}$$

If $\text{quant} : \mathbb{Q} \rightarrow X$ is a function coercing a quantity to a value from X , and X is ordered and has an associated addition and multiplication operation, then we can define the sum of the elements of a bag:

$$\text{sum}(N) \triangleq \sum_{x \in X} x \times \text{quant}(N(x))$$

As in conventional set theory, the concept of ordered pair is introduced, and generalised to tuples of arbitrary length (also denoted as sequences). Sequences can be denoted by enumeration, e.g. $\langle a, b, c, d \rangle$. The operator Lin converts a tuple (of any length) to the corresponding multiset:

$$\text{Lin}(\langle x_1, \dots, x_n \rangle) \triangleq \{ \{ x_1, \dots, x_n \} \}$$

for example $\text{Lin}(\langle a, b, c, d, a \rangle) = \{ \{ a, a, b, c, d \} \}$.

6.2.3 Path Expressions

The syntax of path expressions is presented as an abstract syntax. The semantics of path expressions will be defined using denotational semantics (see e.g. [Sto77], [Wat91]). The semantics of each syntactical construct is either defined in terms of other syntactical constructs, or directly in terms of bags defined in the previous subsection. In denotational semantics, an important role is played by the environment, representing the state of a program. In the case of path expressions, the environment is the population of the information structure. Information descriptors are evaluated in the context of this environment.

As a path expression corresponds to a (directed) path through the information structure diagram, such a path is interpreted as a relation between the object types at its beginning and ending point. Path expressions may be heterogeneous as a result of uniting path expressions with different ending points. In this case, the path expression leads to an heterogeneous binary relation. Consequently, the semantics of path expressions are defined as binary relations over (multiple) object types. Path expressions are built around the following syntactical categories: constant, bag, object type (\mathcal{O}), object type evolution (H_{type}), predicate (\mathcal{P}), and path expression ($\mathcal{PE}(\mathcal{I})$). The naming conventions are: c for constants, X for bags, x for object types, h for object type evolutions, p for predicates and P, Q, G , and P_1, \dots, P_n for path expressions. The function

$$\mu : \mathcal{T} \times \mathcal{PE} \times \text{POP} \rightarrow \Omega_{\mathcal{PE}}$$

where $\text{POP} \triangleq \mathcal{T} \rightarrow (\mathcal{O} \rightarrow \wp(\Omega))$, and $\Omega_{\mathcal{PE}} \triangleq \mathbb{C}(\Omega \times \Omega)$, is used to define the semantics of path expressions. First the *atomic path expressions* are introduced. Note the use of the function Sqr , necessary due to the interpretation of path expressions as binary bag relations.

name	expr	$\mu_t[\llbracket \text{expr} \rrbracket](\text{Pop})$
<i>empty path</i>	$\emptyset_{\mathcal{PE}}$	$0_{\Omega \times \Omega}$
<i>neutral path</i>	$1_{\mathcal{PE}}$	$1_{\Omega \times \Omega}$
<i>constant</i>	c	$\text{Sqr}(\{\llbracket c \rrbracket\})$
<i>bag</i>	X	$\text{Sqr}(X)$
<i>object type</i>	x	$\text{Sqr} \cdot \text{Multi} \cdot \text{Pop}_t(x)$
<i>object type evolution</i>	h	$\text{Sqr} \cdot \text{Multi} \cdot \text{Pop}_t \cdot h(t)$
<i>predicator</i>	p	$\{\llbracket \langle v(p), v \rangle \uparrow^1 \mathbb{Q} \mid v \in \text{Pop}_t \cdot \text{Fact}(p) \rrbracket\}$

A number of operators and functions are available for the construction of composed path expressions. First the unary operators on a path expression (P) are introduced. They provide the opportunity to reverse a path (P^\leftarrow), to isolate the front elements of a path (\mathcal{f}), to remove multiple occurrences (ds), to count the number of elements in a path expression (Cnt), to add the elements in a path expression (Sum), and to determine the minimum or maximum element in a path expression (Min and Max). Further, the power set $\wp(P)$ of a path expression P yields a path expression with all sets of instances occurring in the first component of P , and the $\alpha(P)$ operation selects an arbitrary element from P . The final two unary operations are concerned with temporal aspects. Using $\mathcal{U}(P)$, the periods of time during which a path expression P yields a non empty result can be determined, while the $\mathcal{T}(P)$ operation can be employed to gather information over time. The operators are summarised in the following table:

name	expr	$\mu_t[\llbracket \text{expr} \rrbracket](\text{Pop})$
<i>reverse</i>	P^\leftarrow	$\mu_t[\llbracket P \rrbracket](\text{Pop})^\leftarrow$
<i>front</i>	$\mathcal{f}(P)$	$\text{Sqr} \cdot \pi_1 \cdot \mu_t[\llbracket P \rrbracket](\text{Pop})$
<i>distinct</i>	$\text{ds}(P)$	$\text{ds}(\mu_t[\llbracket P \rrbracket](\text{Pop}))$
<i>count</i>	$\text{Cnt}(P)$	$\text{Sqr}(\{\llbracket \mu_t[\llbracket P \rrbracket](\text{Pop}) \rrbracket\})$
<i>sum</i>	$\text{Sum}(P)$	$\text{Sqr}(\{\llbracket \text{sum} \cdot \pi_1 \cdot \mu_t[\llbracket P \rrbracket](\text{Pop}) \rrbracket\})$
<i>minimum</i>	$\text{Min}(P)$	$\text{Sqr}(\{\llbracket \text{min} \cdot \pi_1 \cdot \mu_t[\llbracket P \rrbracket](\text{Pop}) \rrbracket\})$
<i>maximum</i>	$\text{Max}(P)$	$\text{Sqr}(\{\llbracket \text{max} \cdot \pi_1 \cdot \mu_t[\llbracket P \rrbracket](\text{Pop}) \rrbracket\})$
<i>power set</i>	$\wp(P)$	$\text{Sqr} \cdot \wp \cdot \pi_1 \cdot \mu_t[\llbracket P \rrbracket](\text{Pop})$
<i>pick any</i>	$\alpha(P)$	$\text{Sqr}(\{\llbracket x \uparrow^q \rrbracket\})$ such that: $x \in^q \mu_t[\llbracket P \rrbracket](\text{Pop})$
<i>validity of</i>	$\mathcal{U}(P)$	$\text{Sqr}(\{\llbracket T \uparrow^1 \mathbb{Q} \mid \text{MaxDur}(T, \mu_t[\llbracket P \rrbracket](\text{Pop})) \neq \emptyset_{\mathcal{PE}} \rrbracket\})$
<i>all ever</i>	$\mathcal{T}(P)$	$\bigcup_{t \in \mathcal{T}} \mu_t[\llbracket P \rrbracket](\text{Pop})$

where $\text{MaxDur}(T, P)$ provides the maximal time interval T with respect to predicate P , which is defined as:

1. P holds during T , or: $\forall_{t \in T} [P(t)]$.
2. no larger interval T' has this property: $T' \subseteq T \wedge \forall_{t \in T'} [P(t)] \Rightarrow T = T'$

Example 6.2.1

If $\text{Pop}_t(g) = \{\{r : b_1, s : c_1\}, \{r : b_2, s : \{e_1\}\}, \{r : b_3, s : \{e_2, e_3\}\}\}$ in figure 5.17 (page 91) then:

$$\mu_t \llbracket r \rrbracket (\text{Pop}) = \begin{array}{|c|c|} \hline b_1 & \{r : b_1, s : c_1\} \\ \hline b_2 & \{r : b_2, s : \{e_1\}\} \\ \hline b_3 & \{r : b_3, s : \{e_2, e_3\}\} \\ \hline \end{array}$$

□

Example 6.2.2

In the situation of the previous example:

$$\mu_t \llbracket s^{\leftarrow} \rrbracket (\text{Pop}) = \begin{array}{|c|c|} \hline \{r : b_1, s : c_1\} & c_1 \\ \hline \{r : b_2, s : \{e_1\}\} & \{e_1\} \\ \hline \{r : b_3, s : \{e_2, e_3\}\} & \{e_2, e_3\} \\ \hline \end{array}$$

□

Remark 6.2.1

When applying Elisa-D in an expert system environment, one may also want to associate a probability quantity to instances in the population Pop . This would require a refinement of the definition of HasTypes to:

$$\text{HasTypes} \triangleq (\Omega \times \mathbb{Q}) \times \wp^+(\mathcal{O})$$

and the definition of Pop_t to:

$$\text{Pop}_t(x) \triangleq \bigcup_{v \in \Omega} \{\llbracket v \uparrow^q \rrbracket \mid \langle v, q \rangle \text{ HasTypes}_t Y \wedge x \in Y\}$$

□

A path can be extended in several ways. Path extension by concatenation ($P \circ Q$) is most elementary. The extend operator \diamond also applies to path expressions ($P \diamond Q$), and is built from the head values of both path expressions. Further, the usual set operators ($P \cap Q$, $P \cup Q$ and $P - Q$) are available. These operators are formally described by:

$$\mu_t \llbracket P \Theta Q \rrbracket (\text{Pop}) \triangleq \mu_t \llbracket P \rrbracket (\text{Pop}) \Theta \mu_t \llbracket Q \rrbracket (\text{Pop})$$

for each $\Theta \in \{\circ, \diamond, \cap, \cup, -\}$.

Example 6.2.3

In the situation of example 6.2.1:

$$\mu_t \llbracket r \circ s^{\leftarrow} \rrbracket (\text{Pop}) = \begin{array}{|c|c|} \hline b_1 & c_1 \\ \hline b_2 & \{e_1\} \\ \hline b_3 & \{e_2, e_3\} \\ \hline \end{array}$$

A more complex example making use of the implicit fact type between a power type and its element type is:

$$\mu_t \llbracket r \circ s^{\leftarrow} \circ \in_{F,E}^c \circ \in_{F,E}^e \rrbracket (\text{Pop}) = \begin{array}{|c|c|} \hline b_2 & e_1 \\ \hline b_3 & e_2 \\ \hline b_3 & e_3 \\ \hline \end{array}$$

□

Functions and binary relations can also be used in path expressions.

name	expr	$\mu_t[\llbracket \text{expr} \rrbracket](\text{Pop})$
<i>function application</i>	$f(P_1, \dots, P_n)$	see below
<i>binary relation</i>	$P \text{ R } Q$	see below

If f is an n -ary function, then function application of f on n path expressions yields a path expression where the front elements result from function application on all possible front element combinations of the path expressions involved.

$$\mu_t[\llbracket f(P_1, \dots, P_n) \rrbracket](\text{Pop}) \triangleq \{ \langle f(x_1, \dots, x_n), x \rangle^{\uparrow^k} \mid \forall_{1 \leq i \leq n} [x_i \in \pi_1 \cdot \mu_t[\llbracket P_i \rrbracket](\text{Pop})] \wedge \langle x_n, x \rangle \in^k \mu_t[\llbracket P_n \rrbracket](\text{Pop}) \}$$

Binary functions (operators) will usually be used as infix operations, for instance $P + Q$.

Example 6.2.4

For any population Pop:

$$\mu_t[\llbracket 45 + 25 \rrbracket](\text{Pop}) = \begin{array}{|c|c|} \hline 70 & 25 \\ \hline \end{array}$$

□

If R is a binary relation, then path expression $P \text{ R } Q$ is a concatenation of P and Q via elements of R.

$$\mu_t[\llbracket P \text{ R } Q \rrbracket](\text{Pop}) \triangleq \mu_t[\llbracket P \rrbracket](\text{Pop}) \circ \text{Multi}(\text{R}) \circ \mu_t[\llbracket Q \rrbracket](\text{Pop})$$

Example 6.2.5

Let P and Q be path expressions and Pop a population such that:

$$\begin{aligned} \mu_t[\llbracket P \rrbracket](\text{Pop}) &= \begin{array}{|c|c|} \hline a_1 & 17 \\ a_2 & 19 \\ \hline \end{array} \\ \mu_t[\llbracket Q \rrbracket](\text{Pop}) &= \begin{array}{|c|c|} \hline 16 & b_1 \\ 20 & b_2 \\ 18 & b_3 \\ 18 & b_3 \\ \hline \end{array} \end{aligned}$$

then:

$$\mu_t[\llbracket P < Q \rrbracket](\text{Pop}) = \begin{array}{|c|c|} \hline a_1 & b_2 \\ a_1 & b_3 \\ a_1 & b_3 \\ a_2 & b_2 \\ \hline \end{array}$$

□

Special constructs are available for data type conversions. Grouping and ungrouping form the conversion between an object type and a corresponding power type. Ordering is used for the conversion of a path expression into a sequence. The confluence construct is employed to integrate different sorts of information.

name	expr	$\mu_t[\llbracket \text{expr} \rrbracket](\text{Pop})$
<i>grouping</i>	$\varphi(P, G)$	see below
<i>ungrouping</i>	$\Upsilon(P)$	$\text{Sqr} \cdot \biguplus \cdot \pi_1 \cdot \mu_t[\llbracket P \rrbracket](\text{Pop})$
<i>ordering</i>	$\psi(P, G)$	see below
<i>confluence</i>	$[P_1, \dots, P_n \mid Q]$	see below

Grouping path expression P , according to grouping criterion G , is performed by the function $\varphi(P, G)$. The elements to be grouped are obtained from the first component of path expression P . Path expression G specifies a grouping criterion for these elements. If $g \in \pi_2 \cdot \mu_t[G](\text{Pop})$, then with g the following class of elements is associated:

$$K_g \triangleq \{x \in \pi_1 \cdot \mu_t[P](\text{Pop}) \mid \langle x, g \rangle \in \mu_t[G](\text{Pop})\}$$

The result of grouping is now obtained as the set of all such classes, presented in the format that is used for the interpretation of path expressions:

$$\mu_t[\varphi(P, G)](\text{Pop}) \triangleq \{\langle K_g, g \rangle^{\uparrow 1_Q} \mid g \in \pi_2 \cdot \mu_t[G](\text{Pop}) \wedge K_g \neq \emptyset\}$$

Sorting the result of path expression P into a single sequence, according to a sorting criterion S , can be achieved by applying ψ on P and S respectively. The sorting criterion may be weak (for example $S = \emptyset_{\mathcal{PE}}$), allowing more than one ordering of the elements, or too strong, for which any ordering fails. A sequence s is called compatible with sorting criterion S over P in population Pop_t if:

1. s contains all elements of $\pi_1 \cdot \mu_t[P](\text{Pop})$ in the same frequency:

$$\text{Lin}(s) = \pi_1 \cdot \mu_t[P](\text{Pop})$$

2. the order of elements in s does not conflict with the ordering rules from S , i.e. for each i, j such that $0 \leq i < j < |s|$:

$$\exists_{y_1, y_2} [\langle s_{<i>, y_1} \rangle \in \mu_t[P](\text{Pop}) \wedge \langle s_{<j>, y_2} \rangle \in \mu_t[P](\text{Pop}) \wedge \langle y_2, y_1 \rangle \notin \mu_t[S](\text{Pop})]$$

The result of sorting is defined as:

$$\mu_t[\psi(P, S)](\text{Pop}) \triangleq \text{Sqr}(\{\{s^{\uparrow 1} \mid s \text{ is compatible with } S \text{ over } P \text{ in } \text{Pop}_t\}\})$$

The confluence operation is a powerful operation on path expressions. This operator is typically used when different sorts of information are to be integrated. For instance, name, day of birth, salary and address of an employee with a given employee number. If P_1, \dots, P_n, Q are path expressions then $[P_1, \dots, P_n \mid Q]$ is a path expression corresponding to an n -ary relation called the confluence of P_1, \dots, P_n with condition Q . The meaning of this expression is:

$$\begin{aligned} & \mu_t[P_1, \dots, P_n \mid Q](\text{Pop}) \\ & \triangleq \bigcup_{x \in \pi_1 \cdot \mu_t[Q](\text{Pop})} \{\langle \langle x_1, \dots, x_n \rangle, x \rangle^{\uparrow k_1 \times \dots \times k_n} \mid \forall_{1 \leq i \leq n} [\langle x_i, x \rangle \in^{k_i} \mu_t[P_i](\text{Pop})]\} \end{aligned}$$

The condition in the confluence Q , is not mandatory. By using $1_{\mathcal{PE}}$, the condition is neutralised. As a shorthand, we therefore define: $[P_1, \dots, P_n] \triangleq [P_1, \dots, P_n \mid 1_{\mathcal{PE}}]$.

To define the active complement \neg (see [Mai88]) of a path expression, the set of active elements are introduced:

$$\text{ActVals} \triangleq \bigcup_{x \in \mathcal{O}} x$$

The *active complement* of a path expression P then, is defined as: $\neg P \triangleq \text{ActVals} - f P$.

A set, or sequence, of path expressions can be converted to a path expression consisting of sets, or sequences, of ‘ordinary’ elements. This is achieved by the set and sequence constructor.

name	expr	$\mu_t[\text{expr}](\text{Pop})$
sequence constructor	$\langle P_1, \dots, P_n \rangle$	$\text{Sqr} \cdot \text{Multi}(\{\langle x_1, \dots, x_n \rangle \mid \forall_{1 \leq i \leq n} [x_i \in \pi_1 \cdot \mu_t[P_i](\text{Pop})]\})$
set constructor	$\{P_1, \dots, P_n\}$	$\text{Sqr} \cdot \text{Multi}(\{\{x_1, \dots, x_n\} \mid \forall_{1 \leq i \leq n} [x_i \in \pi_1 \cdot \mu_t[P_i](\text{Pop})]\})$

Usually the path expressions P_1, \dots, P_n in the set and sequence constructor will contain just one value.

The schema constructor allows for the definition of a path expression which is a denotation of a function of object types to path expressions. This constructor is used for the denotation of instances from schema types. In these cases, the path expressions involved (possibly zero) contain at most one element.

name	expr	$\mu_t[\text{expr}] (\text{Pop})$
<i>schema construction</i>	$\text{sc}(X_1 : P_1, \dots, X_n : P_n)$	$\text{Sqr}(\{\langle X_1, y_1 \rangle, \dots, \langle X_n, y_n \rangle\})$ where $y_i \in \pi_1 \cdot \mu_t[P_i] (\text{Pop})$ for each $1 \leq i \leq n$

Note that the P_i in the above definition should be path expressions resulting in sets.

When using object types from the decomposition of a schema type in a path expression, the need arises to restrict the populations of these object types to *one* instance of the schema type. For this purpose, we define the following operation for any schema type s :

name	expr	$\mu_t[\text{expr}] (\text{Pop})$
<i>select population</i>	$\sigma(P, s, Q)$	$\bigcup_{\langle p, q \rangle \in \mu_t[s \circ Q] (\text{Pop})} \mu_t[P] (S(\text{Pop}, s, p))$

where the function $S : \text{POP} \times \mathcal{C} \times \text{POP} \rightarrow \text{POP}$ is identified by:

$$S(\text{Pop}^1, s, \text{Pop}^2) \triangleq \lambda \langle t, x \rangle . \text{if } s \prec^* x \text{ then } \text{Pop}_t^2(x) \text{ else } \text{Pop}_t^1(x) \text{ fi}$$

We introduce two mechanisms in Elisa-D for the introduction of variables, a macro mechanism resulting in a *call by name* variable mechanism, and a *call by value* variable mechanism (see e.g. [ASU86]). To accommodate the latter class of variables, we introduce the *freeze operator* \dagger , which operates on elements of $\Omega_{\mathcal{PE}}$. Evaluation of this operator applied to an element $x \in \Omega_{\mathcal{PE}}$ simply yields this element, which will be the *value* stored in the call by value variable.

Macros in Elisa-D can be recursive. Therefore, we apply lazy evaluation for the evaluation of macros. The ϵ operation are used to evaluate macros in the context of the actual population. Macros will not be evaluated until the path expression, in which the macro is contained, is evaluated. On the path expression level, a macro call therefore corresponds to the expression $\epsilon(M)$ where M denotes the macro to be evaluated. A macro, then corresponds to a function: $M : \mathcal{T} \times \text{Pop} \rightarrow \Omega_{\mathcal{PE}}$. On the Elisa-D level, a macro will be translated to a corresponding function (e.g. M) on the path expression level. The two variable mechanisms discussed lead to the following definitions:

name	expr	$\mu_t[\text{expr}] (\text{Pop})$
<i>freeze operator</i>	$\dagger(x)$	x
<i>evaluate macro</i>	$\epsilon(M)$	$M_t(\text{Pop})$

Finally, for path expressions we also define a comprehension schema, and a conditional evaluation construction:

name	expr	$\mu_t[\text{expr}] (\text{Pop})$
<i>comprehension schema</i>	$\{v \in P \mid C\}$	see below
<i>conditional clause</i>	if C then P else Q	$\begin{cases} \mu_t[P] (\text{Pop}) & \text{if } \mu_t[C] (\text{Pop}) \neq \emptyset_{\mathcal{PE}} \\ \mu_t[Q] (\text{Pop}) & \text{otherwise} \end{cases}$

The semantics of the comprehension schema is provided as:

$$\mu_t[\{v \in P \mid C\}] (\text{Pop}) \triangleq \{\langle x, y \rangle \uparrow^q \mid \langle x, y \rangle \in^q \mu_t[P] (\text{Pop}) \wedge \mu_t[C[v := x]] (\text{Pop}) \neq \emptyset_{\mathcal{PE}}\}$$

The expression $C[v := x]$ represents the path expression C , in which all occurrences of name v are interpreted as constant x (note that x is a meta variable, but a constant in the semantics of C).

Remark 6.2.2

In the context of a document retrieval system, path expressions can easily be extended with a relevance feedback mechanism. Suppose every value of the population (in this case documents) has an associated characterisation: $\chi : \Omega \rightarrow \mathcal{C}$. Such a characterisation can be a set of keywords, or an index expression defined in [Bru93].

In document retrieval systems, the user will specify an information need q , which will then be matched with these characterisations. The result of this match, provides the relevance of the associated information object to the user.

When using a suitable calculation domain, one can define the operation $\text{Rel}(q)$ returning a path expression containing information objects (from Ω), and their relevance with respect to information need q .

Such a seamless integration between a query language, and a language for the retrieval of documents, would allow for queries such as:

List the authors and publishers of books on “air pollution”

□

6.3 Deriving the Disclosure Schema

A most important aspect of an information system, is to support information retrieval. It would indeed not make sense to store information, if the stored information will never be retrieved. Retrieving information from an evolving information system is inherently richer than retrieving information from a traditional information system, due to the availability of a time axis for both the population and the schema. Therefore, a language supporting evolving information systems, should provide the opportunity to query:

- the current state of the system (both population and structure)
- previous states of the system (both population and structure), and their relations

We provide a synergy of the several aspects at three levels of abstraction, leading to the *disclosure schema*. The disclosure schema is the point of departure for the introduction of the way of communicating for an evolving information system.

6.3.1 The extra-temporal schema

In this section we start by introducing the *extra-temporal schema* of an application model. The extra-temporal information structure is defined as the union of all information structure versions described by application model history $H: \mathcal{O}_\infty \triangleq \bigcup_{t \in \mathcal{T}} \mathcal{O}_t$.

As an illustration of an extra-temporal information structure, consider the two schema versions in figure 6.2, the evolution from a binary fact type (f) to a ternary fact type (g) is modelled in this figure. The associated extra-temporal information structure is depicted in figure 6.3.

Constraints associated with versions of the application model, will not necessarily hold for the extra-temporal population, for evolutionary information systems constraints must be enforced in their proper temporal context, and because of this, the extra-temporal schema is the same as the extra-temporal information structure.

6.3.2 The disclosure schema

In an evolving information system, the user may want to query the past and present of the application model as a whole. Querying is not limited to the population, but may also involve such things as the information structure and

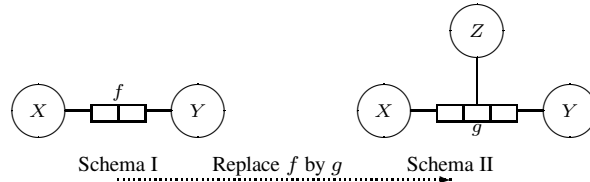


Figure 6.2: Evolution of schemas

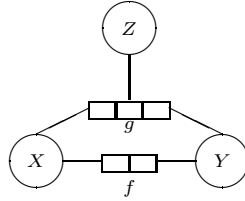


Figure 6.3: A simple extra-temporal schema

activity specifications. We approach this problem by introducing a disclosure schema, comprising the extra-temporal schema, and the meta schema of the modelling techniques used. The disclosure schema will be defined gradually via a number of subschemas (stepwise refinement). The overall meta schema of the evolution theory was provided in figure 3.10 (page 55).

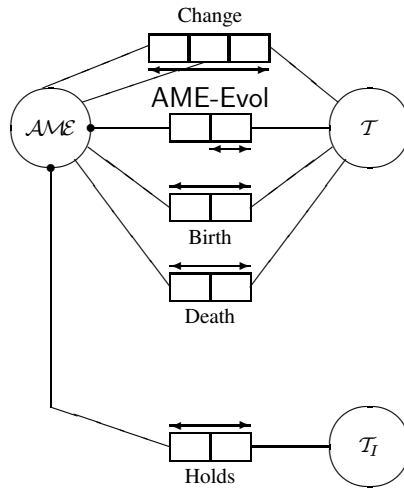


Figure 6.4: Extension with temporal relations

As a first refinement, the schema is augmented with the meta model for updates in evolving information systems, depicted in figure 4.11 (page 77). As a second refinement, the schema is augmented with a set of (derivable) relations, enabling a convenient formulation of queries involving time (see figure 6.4). For example, the fact type *Change* keeps track of all changes undergone by histories $h \in H$ as triples $\langle h(t), h(\triangleright t), t \rangle$. A change occurs at time t in history h , if h is alive at points in time t and $\triangleright t$, but in different shapes:

$$\text{Pop}_t(\text{Change}) \triangleq \{ \{ c.fe:h(t), c.te:h(\triangleright t), c.t:t \} \mid h \in H \wedge h \downarrow t, \triangleright t \wedge h(t) \neq h(\triangleright t) \}$$

where $c.fe$ (from element), $c.te$ (to element) and $c.t$ (time) are the predicates from the fact type *Change*. The population of the fact types *Birth* and *Death* can be derived analogously:

$$\begin{aligned} \text{Pop}_t(\text{Death}) &\triangleq \{ \{ d.e:h(t), d.t:t \} \mid h \in H \wedge h \downarrow t \wedge h \uparrow \triangleright t \} \\ \text{Pop}_t(\text{Birth}) &\triangleq \{ \{ b.e:h(\triangleright t), b.t:\triangleright t \} \mid h \in H \wedge h \uparrow t \wedge h \downarrow \triangleright t \} \end{aligned}$$

Fact type **Holds** relates application model elements to time intervals as follows. Each version $h(t)$ of application model evolution h is related to the (maximal) time interval T during which this version holds. The population of **Holds** is defined as:

$$\text{Pop}_t(\text{Holds}) \triangleq \{ \{ h.e : h(t), h.ti : T \} \mid h \in H \wedge h \downarrow t \wedge \text{MaxDur}(T, P) \}$$

where P is defined by: $P(s) \triangleq \exists_{g \in H} [h(t) = g(s)]$.

More relations can be defined on \mathcal{T} and \mathcal{T}_I . Three classes of temporal relations can be identified ([RP92]). Relations over \mathcal{T} , such as **BEFORE**, and **AFTER**, capture comparison relations for points in time, and form the first class. Two comparison relations for points in time, which are based on \triangleright , can be defined as:

$$t_1 \text{ HAS INCREMENT } t_2 \iff \triangleright t_1 = t_2$$

$$t_1 \text{ IS INCREMENT OF } t_2 \iff t_1 = \triangleright t_2$$

The second class of comparison relations deals with time intervals (\mathcal{T}_I), and includes relations such as: **BEFORE**, and **OVERLAPPED-BY**. Comparing points in time with time intervals is the objective of the third class, and contains for example: **CONTAINS**. The semantics of these relations, depends on the time axis chosen \mathcal{T} , and will not be elaborated upon here. As an overview of possible temporal relations, consider figure 6.5, which is based on [RP92].

Next, the meta schema is refined for the EVORM data modelling technique, by extending it with the EVORM meta schema as depicted in figure 5.13. Doing so, leads to a refined schema which is more specific about the $\langle \mathcal{O}, \mathcal{N}, \mathcal{L} \rangle$ triplet and the filtering of the **ldfBy**-relation, thus providing a technique dependent extension of the general meta model. When a concrete modelling technique has been chosen for the modelling of constraints and methods, the meta model of the general theory can also be augmented with more details on constraints and activities.

On the one hand, the extra-temporal schema of an application model evolution forms a population of the $\langle \mathcal{O}, \mathcal{N}, \mathcal{L} \rangle$ (meta) triplet. On the other hand, the extra-temporal schema is a subschema of the disclosure schema. In figure 6.6, the connection between the extra-temporal schema of figure 6.3 and the meta schema is provided as a sub schema, expressing this dual relationship. In general, the **Gen** relation of the disclosure schema is extended with $x \text{ Gen } \Omega$ for all object types $x \in \mathcal{O}_\infty$.

6.3.3 Three levels of abstraction

In the above discussion, leading to the disclosure schema, three levels can be distinguished (see figure 6.7). The *instance level* deals with the extra-temporal population. The extra-temporal schema forms the second level, the *schema level*. The top level, the *meta schema level*, completes the disclosure schema, and adds the possibility of looking beyond temporal boundaries, by taking temporal relations into account.

The instance level is related to the schema level through the **HasTypes** relation, while the relation between the schema level and the meta schema level is formed by the generalisation relation.

The three levels of abstraction distinguished, may be regarded as three layers in a *multi-layered hypermedia system* ([BW92b]). This observation enables the disclosure of the information contained in an evolving information system using an hypertext approach. This approach has been studied before in [BPW93], and is essential for good information disclosure in evolving information systems, as the underlying information structure (second level) changes regularly. We will elaborate upon this in chapter 8.

6.4 Information Descriptors

The NIAM analysis method uses a natural language approach. The method starts from verbalisations of examples, which form a (partial) description of the underlying domain, and are provided by domain experts. We refer to the language (idiom), in which the examples are verbalised, as the expert language. This verbalisation leads eventually to the information structure of the universe of discourse under consideration. For more details, see [NH89], [CW93].

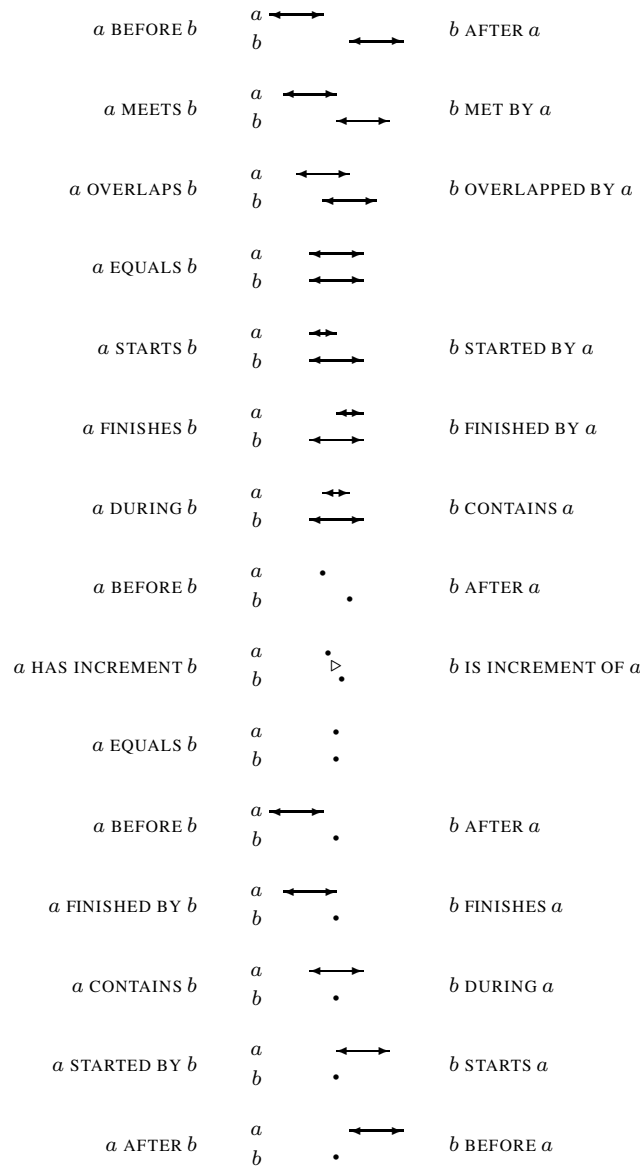


Figure 6.5: A suite of temporal relations

It is desirable that the language for manipulating and querying also has the format of a semi-natural language, and is designed to approximate the expert language as close as possible (suitability, see L2 (page 102)). The rationale behind this has been addressed in [HH93]. As a result, sentences in this (artificial) language are meaningful expressions within the context of the universe of discourse, understandable and expressible by domain experts (contributing to elegance, see L3). Sentences, verbalising the original examples, form extensional specifications, while queries (in general) correspond to intensional specifications. Such a language, Elisa-D, based on the data modelling technique EVORM is presented in this section. Elisa-D is the evolution supporting pendant of LISA-D ([HPW93]). As EVORM generalises object-role modelling techniques (such as EER, ER, NIAM and FORM), Elisa-D is generally applicable.

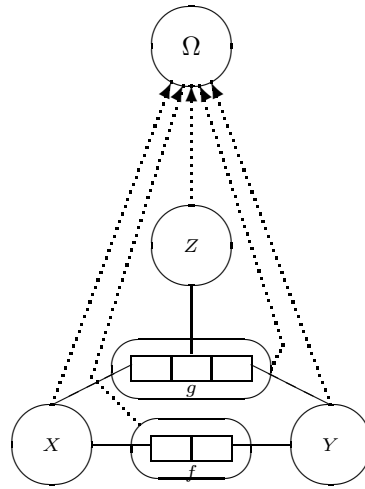


Figure 6.6: Extra-temporal schema linked to meta schema

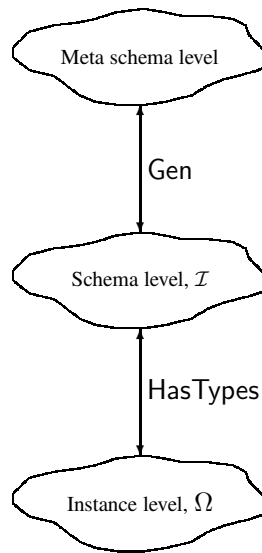


Figure 6.7: Three levels of abstraction

6.4.1 The structure of Elisa-D

In the previous sections, the elements constituting an application model were introduced as abstract concepts (the way of modelling). The intention of this section is to describe a language (way of communicating) by which the universes of discourse's walk of life can be communicated (by human beings) to the information system (see figure 1.11). This language is set up as a (semi-) *natural language*, resembling the expert language.

Typical for a natural language is the richness to form sentences, even sentences that have no intuitive meaning, or sentences that are ambiguous. Rather than excluding senseless information descriptors syntactically, the semantic interpretation will yield a void for such constructs. Static semantics checks can easily detect such flaws in information descriptors. The language should be such that it allows for an elegant description of a user's information need (requirement L3). This does not imply the exclusion of inelegant descriptions, though ideas of what is elegant can differ!

Elisa-D provides a set of initial grammar rules, that when complemented by a concrete *lexicon*, as obtained from

a particular application model, results in a concrete information retrieval and update language. This is analogous to, say, predicate calculus, in which a set of predicate symbols provides the lexicon, and the construction rules for formulae correspond to the grammar. The concrete language can be further tuned to the application domain, by extending the grammar with new rules. (Note that conventional programming languages use the procedure or macro mechanism for this purpose.) As a result, Elisa-D defines a class of languages, where each language is based on a particular underlying lexicon, and a set of grammar rules. We will describe the naming conventions used and provide the semantics of these names in one of the next subsections.

Most of the examples in this section are taken from a fragment of the *Presidential Database*, regarding the process of electing presidents in the USA. This example is a unified example given in the special issue of Computing Surveys ([FS76]). The example was first used in [WBGW73]. An excerpt of this schema is presented in figure 6.8. Note that we have omitted the constraints for reasons of clarity.

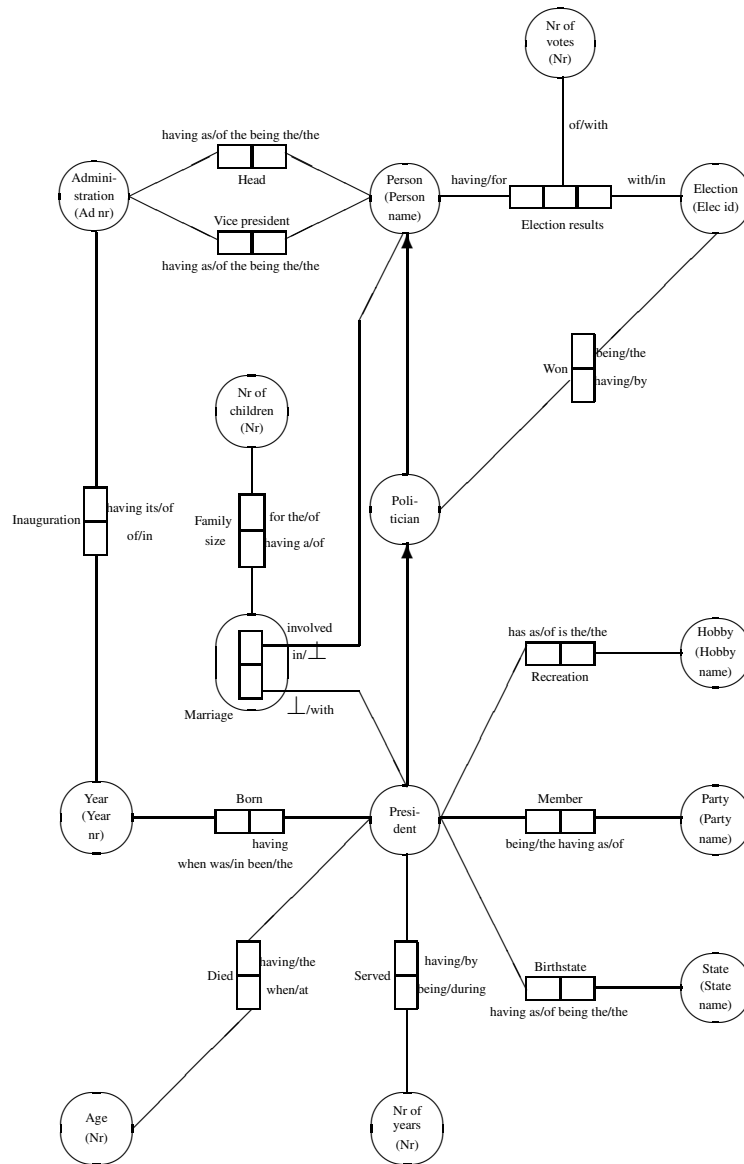


Figure 6.8: Part of an information structure describing American presidents

Abstract name	Concrete name	Abstract name	Concrete name
\mathcal{T}	Time	\mathcal{D}	Domain
\mathcal{AMH}	Amh	Dom	Domain assignment
AME-Evol	Ame evolution
\mathcal{AME}	Ame	\mathcal{AMEV}	Am evolution
\mathbb{I}	Evolution step	\mathcal{AMRH}	Am recording history
\mathcal{EO}	Event occurrence	$\mathcal{T} \mapsto \mathcal{T}$	Recording time
\mathcal{M}	Methods	\mathcal{T}_I	Time interval
\mathcal{O}	Object type	Holds	Holds
$\mathcal{O}(\mathcal{O})$	Types	Change	Change
\mathcal{N}	Non label type
\mathcal{L}	Label type	\mathcal{E}	Entity type
\leadsto	Parent	\mathcal{F}	Fact type
\sim	Type related	\mathcal{G}	Power type
Ω	Value	\mathcal{P}	Predicator
HasTypes	Instance typing

Table 6.1: Names for meta object types

6.4.2 The assignment of names

The basis for Elisa-D information descriptors is a lexicon, assigning a meaning to the words (names) that constitute the language. The meaning of names is administered by the partial naming function:

$$\text{Lexicon} \subseteq \text{Names} \times \mathcal{PE}$$

If $\text{Lexicon}(n, P)$, then the name n can be used as a denotation for an instance evolution described by P . In this case, name n is qualified as a *defined name*.

The function Lexicon is filled with a number of predefined names, and may be extended in the course of time by the user of an Elisa-D interpreter. A first class of names verbalises the highest level of abstraction (see figure 6.7), and provides names for the meta concepts at this level. These names are provided by the function $\text{ONm} : \mathcal{O}^M \mapsto \text{Names}$, where \mathcal{O}^M is the set of all object types in the meta schema. This function is summarised in table 6.1. The meta level consists of:

1. the meta schema for the general evolution theory, see figure 3.10 (page 55),
2. the meta schema for updates in evolving information systems, see figure 4.11 (page 77),
3. the additional temporal aspects as depicted in figure 6.4 (page 113),
4. the EVORM-specific details, contained in figure 5.13 (page 87).

The second class of names covers the extra-temporal schema level, and provides names for application specific concepts:

1. names for element evolutions are recorded in an injective function, and provide unique names: $\text{HNm} : \mathcal{AMH} \mapsto \text{Names}$,
2. object types (\mathcal{O}) are referenced by a name provided by the function ONm (Note that this function is also used for the naming of meta object types),
3. names of domains are found by the injective function: $\text{Dnm} : \mathcal{D} \mapsto \text{Names}$.

The names of object types should be unique at any point in time:

$$\text{ONm}(x) = \text{ONm}(y) \wedge x, y \in \mathcal{O}_t \Rightarrow x = y$$

Multiple object types with the same name, are allowed to exist in different versions of the information structure.

The naming of instances (Ω) consists of a denotation mechanism for label values (constants), and a naming mechanism for abstract values. This latter mechanism is modelling technique dependent. In EVORM, the identification mechanism, leading to standard names, is used for this purpose.

Predicators may have assigned a *predicator name* via the (partial) function: $\text{PNm} : \mathcal{P} \rightarrow \text{Names}$, and a *predicator reverse name*, via $\text{RNm} : \mathcal{P} \rightarrow \text{Names}$. In figure 6.8, the predictors have associated a combined predicator name p and reverse predicator name r , in the format p/r .

So far, the fact types in example EVORM schemas (e.g. figure 5.9 (page 84)) have been labeled with connector names. Connector names (also referred to as role names), will be used to verbalise the transition through a fact type via two (which may be the same) of its predictors. In section 6.5 we provide a macro mechanism allowing for the enrichment of the language. There, we will use connector names as examples to improve the readability of Elisa-D expressions.

A predicator name should be unique in the context of a fact type:

$$\text{Fact}(p) = \text{Fact}(q) \wedge \text{PNm}(p) = \text{PNm}(q) \Rightarrow p = q$$

$$\text{Fact}(p) = \text{Fact}(q) \wedge \text{RNm}(p) = \text{RNm}(q) \Rightarrow p = q$$

This requirement allows for the following operators, retrieving the predicator associated with a predicator name within a fact type:

$$\text{Pred} : \text{Names} \times \text{Names} \rightarrow \mathcal{P} \text{ and } \text{RPred} : \text{Names} \times \text{Names} \rightarrow \mathcal{P}$$

which are identified as:

$$\forall_{p \in \mathcal{P}} [\text{Pred}(\text{ONm} \cdot \text{Fact}(p), \text{PNm}(p)) = p] \text{ and } \forall_{p \in \mathcal{P}} [\text{RPred}(\text{ONm} \cdot \text{Fact}(p), \text{RNm}(p)) = p]$$

We are now in a position to fill the lexicon with the predefined names and their meaning.

6.4.2.1 Named concepts

The name function Lexicon contains a verbalisation of the concepts in the disclosure schema. Here we provide an initial filling of the Lexicon function.

If h is an object type history with name $\text{HNm}(h)$, then $\text{HNm}(h)$ is a defined name:

$$\text{Lexicon}(\text{HNm}(h), h)$$

As stated in chapter 5, EVORM distinguishes between implicit and explicit object types. No names are assumed for implicit object types (such as fact type \in_x), special names will be introduced later to handle their manipulation. Let x be an explicit object type, then the name $\text{ONm}(x)$ refers to the path expression x :

$$\text{Lexicon}(\text{ONm}(x), x)$$

If p is a named predicator, then the name $\text{PNm}(p)$ describes a path from the base of p to its corresponding fact type, while $\text{RNm}(p)$ describes the reverse path:

$$\text{Lexicon}(\text{PNm}(p), p) \wedge \text{Lexicon}(\text{RNm}(p), p^{\leftarrow})$$

Defined names for label values, make it possible to use such denotations as regular information descriptors. If c is a constant and x an object type such that $c \in \text{Dom}_t(x)$, then the denotation $\text{CNm}(c)$ may be interpreted according to:

$$\text{Lexicon}(\text{CNm}(c), c)$$

6.4.2.2 Anonymous concepts

A schema (and thus the disclosure schema) may contain a number of implicit fact types; being implicit, such fact types, and their constituent predicators, are not named. It may, however, be necessary to address these concepts in verbalisations of information needs. Therefore, a set of default names for these concepts is introduced (see figure 6.9).

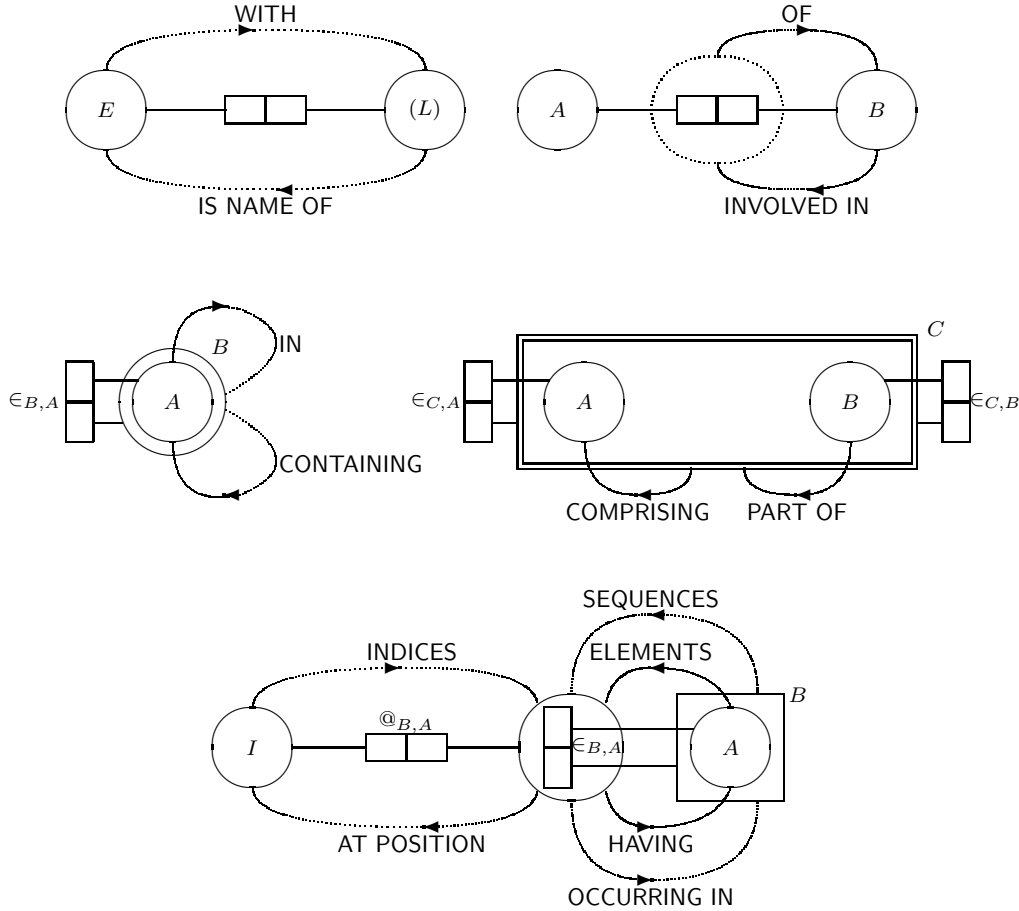


Figure 6.9: Names of implicit fact types

The names WITH and IS NAME OF can be used for quickly relating object types to label types. The name WITH relates object types via bridge types to label types, the name IS NAME OF being its reverse. Let $b = \{p, q\}$ be a bridge type, relating non-label type $\text{Base}(p)$ to label type $\text{Base}(q)$, then:

$$\text{Lexicon}(\text{WITH}, p \circ b \circ q^{\leftarrow}) \wedge \text{Lexicon}(\text{IS NAME OF}, q \circ b \circ p^{\leftarrow})$$

The names CONTAINING and IN verbalise the implicit relation between a power type and its underlying element type. CONTAINING relates a power type to its corresponding element type, IN is its inverse. For all $x \in \mathcal{G}$:

$$\begin{aligned} &\text{Lexicon}(\text{CONTAINING}, \in_{x, \text{Elt}(x)}^c \circ \in_{x, \text{Elt}(x)} \circ \in_{x, \text{Elt}(x)}^e)^{\leftarrow} \wedge \\ &\text{Lexicon}(\text{IN}, \in_{x, \text{Elt}(x)}^e \circ \in_{x, \text{Elt}(x)} \circ \in_{x, \text{Elt}(x)}^c)^{\leftarrow} \end{aligned}$$

The implicit fact types for sequence types describe the indexing mechanism. For all $x \in \mathcal{S}$:

$$\begin{aligned} & \text{Lexicon}(\text{SEQUENCES}, \in_{x, \text{Elt}(x)}^c) \wedge \text{Lexicon}(\text{OCCURRING IN}, \in_{x, \text{Elt}(x)}^c \leftarrow) \wedge \\ & \text{Lexicon}(\text{ELEMENTS}, \in_{x, \text{Elt}(x)}^e) \wedge \text{Lexicon}(\text{HAVING}, \in_{x, \text{Elt}(x)}^e \leftarrow) \wedge \\ & \text{Lexicon}(\text{INDICES}, @_{x, \text{Elt}(x)}^i \circ @_{x, \text{Elt}(x)} \circ @_{x, \text{Elt}(x)}^s \leftarrow) \wedge \\ & \text{Lexicon}(\text{AT POSITION}, @_{x, \text{Elt}(x)}^s \circ @_{x, \text{Elt}(x)} \circ @_{x, \text{Elt}(x)}^i \leftarrow) \end{aligned}$$

The relations between a schema type, and the object types from its decomposition are handled by the keywords **COMPRISING** and **PART OF**. **COMPRISING** relates each instance of a schema type to the instances of the object types of its decomposition. **PART OF** is the reverse of **COMPRISING**. For all $x \in \mathcal{C}, x \prec y$:

$$\text{Lexicon}(\text{COMPRISING}, \in_{x, y}^e \circ \in_{x, y} \circ \in_{x, y}^c \leftarrow) \wedge \text{Lexicon}(\text{PART OF}, \in_{x, y}^c \circ \in_{x, y} \circ \in_{x, y}^e \leftarrow)$$

6.4.2.3 Generic names

The names **OF** and **INVOLVED IN** are intended to facilitate the handling of objectification, they are also useful as shorthands for predicator names. **OF** represents all relations between fact type instances and their constituent object type instances, **INVOLVED IN** being its inverse. For all $q \in \mathcal{P}$:

$$\text{Lexicon}(\text{INVOLVED IN}, q) \wedge \text{Lexicon}(\text{OF}, q \leftarrow)$$

The union operator in this definition is required to handle fact types that contain predicators with the same base.

6.4.3 Syntax and semantics of information descriptors

The main concept in Elisa-D is the *information descriptor*, by which information needs can be formulated. Information descriptors are provided by an abstract syntax, however, in some examples we will resort to concrete syntax to avoid ambiguities.

An information descriptor is interpreted as a special history, describing the evolution of the result of this descriptor over the course of time. At each moment, this result comprises a binary bag. Elisa-D is built around a number of syntactical categories. The **Information Descriptors** form the main category. Updates and queries will follow in later sections. The underlying elementary syntactical categories are: **Var** for simple variables, **Names** for names, **Macro Header** for macro headers (also call by name variables). The naming conventions for instances of these syntactical categories are as follows:

$$\begin{array}{ll} \text{Information Descriptor:} & P, P', P_1, P_2, O, Q \\ \text{Var:} & v, x_1, \dots, x_n, y_1, \dots, y_n \\ \text{Names:} & n, \omega_1, \dots, \omega_n \end{array}$$

This subsection deals mainly with the **Information Descriptor** category, and also introduces the category **Assignment** covering the assignment of values to variables and macro definitions.

The semantics of the syntactic category **Information Descriptor** is specified in terms of path expressions, by the valuation function:

$$\mathbb{D} : \text{Information Descriptor} \times \text{ENV} \rightarrow \mathcal{PE}$$

mapping information descriptors on path expressions. The aim of this function, is to provide a syntactic transformation. Temporal issues are *hidden* in the path expression level. The assignments are introduced by the valuation function:

$$\mathbb{A} : \mathcal{T} \times \text{Assignment} \times \text{ENV} \times \text{POP} \rightarrow \text{ENV}$$

An information descriptor, as is an assignment, is evaluated in the context of an environment. The environment ENV contains the current values of macro definitions and (call by value) variables. The macro mechanism, to be introduced, is defined as a call by name mechanism, the variables are introduced as a call by value mechanism. The domain for environments is therefore defined as:

$$\text{ENV} \triangleq (\mathcal{T} \times \text{Macro Header} \rightarrow \text{Information Descriptor}) \times (\text{Var} \rightarrow \mathbb{C}(\Omega, \mathcal{PE}))$$

The valuation functions are denoted as:

$$\mathbb{D}[\text{expression}] (\langle M, V \rangle) = P, \text{ and } \mathbb{A}_t[\text{expression}] (\langle M, V \rangle, \text{Pop}) = \langle M', V' \rangle.$$

6.4.3.1 Atomic information descriptors

The foundation of information descriptors in Elisa-D is formed by the defined names in Names, introduced in the previous section, and the denotation of (complex) constants.

The meaning of a name is obtained as the sum of all possible interpretations recorded by the Lexicon-function. Variables form another elementary construct for information descriptors as they are used to store intermediate results. The meaning of the elementary constructs is summarised by:

$$\begin{aligned} \mathbb{D}[n] (\langle M, V \rangle) &\triangleq \bigcup_{\text{Lexicon}(n, P)} P \\ \mathbb{D}[v] (\langle M, V \rangle) &\triangleq \begin{cases} V(v) & \text{if } V \downarrow v \\ \emptyset_{\mathcal{PE}} & \text{otherwise} \end{cases} \end{aligned}$$

A variable V is assigned a value, being the result of an information descriptor P at a given point in time t , using the ASSIGN P TO v construction. This construction provides the first rule for Assignments:

$$\mathbb{A}_t[\text{ASSIGN } P \text{ TO } v] (\langle M, V \rangle, \text{Pop}) \triangleq \langle M, V \odot v : \mu_t \cdot \mathbb{D}[P] (\langle M, V \rangle, \text{Pop}) \rangle$$

The use, and definition, of macro definitions in M , is addressed in a later subsection.

Instances from schema types, sequence types and power types are denoted in terms of their constituent elements. These denotations are introduced in Elisa-D by:

$$\begin{aligned} \mathbb{D}[\{P_1, \dots, P_n\}] (e) &\triangleq \{\mathbb{D}[P_1] (e), \dots, \mathbb{D}[P_n] (e)\} \\ \mathbb{D}[\langle P_1, \dots, P_n \rangle] (e) &\triangleq \langle \mathbb{D}[P_1] (e), \dots, \mathbb{D}[P_n] (e) \rangle \\ \mathbb{D}[X_1 \rightarrow P_1, \dots, X_n \rightarrow P_n] (e) &\triangleq \text{sc}(x_1 : \mathbb{D}[P_1] (e), \dots, x_n : \mathbb{D}[P_n] (e)) \end{aligned}$$

where $\text{ONm}(x_i) = X_i$ for all $1 \leq i \leq n$.

Some examples of atomic information descriptors are constant denotations of label types (for example 'Roosevelt F.D.'), names for object types (Year), and predicate names (having its).

6.4.3.2 Juxtaposition of information descriptors

A number of operators are available to construct (composed) information descriptors. The most important construction mechanism is juxtaposition. The general rule for concatenating information descriptors P and Q is:

$$\mathbb{D}[P Q] (e) \triangleq \mathbb{D}[P] (e) \circ \mathbb{D}[Q] (e)$$

A crucial effect of the concatenation operator is that it filters out the apparent intention of the user. Both information descriptors P and Q may be very ambiguous, if they are used in the context of each other, much of the ambiguity will disappear. The strongest case is when both information descriptors have no meaning in each others context, i.e. when there is no connection from the one to the other. If there is no connection between information descriptors,

concatenation will result in an information descriptor with a void meaning:

$$\begin{aligned}
 & \mathbb{D}[\text{President having been Born in Hobby}](e) \\
 & \equiv \{elaborate\} \\
 & \text{Obj(President)} \circ \text{Pred(Born, having been)} \circ \text{Obj(Born)} \circ \\
 & (\text{RPred(Born, in)}^{\leftarrow} \cup \text{RPred(Election results, in)}^{\leftarrow} \cup \text{RPred(Inauguration, in)}^{\leftarrow}) \circ \\
 & \text{Obj(Hobby)} \\
 & \equiv \left\{ \begin{array}{l} \text{The bases of RPred(Born, in), RPred(Election results, in),} \\ \text{and RPred(Inauguration, in) are not type related with Obj(Hobby)} \end{array} \right\} \\
 & \emptyset_{PE}
 \end{aligned}$$

Note that it can be decided statically (i.e. without the need for evaluation) whether a connection exists between two information descriptors. This leads to two filter properties, which can be found in [HPW93].

Some other examples, for the schemas depicted in figure 5.6 (page 83) and figure 5.9 (page 84), of composed information descriptors using juxtaposition are:

$$\{\text{Ship WITH Ship code 'F101', Ship WITH Ship code 'F103'}\}$$

and

$$\text{Freight car sequence COMPRISING Freight car WITH Fc code 'RIV901'}$$

6.4.3.3 Composition of information descriptors

Information descriptors can be combined into new information descriptors by a number of operators. The first group contains operators such as union, intersection and set difference:

$$\begin{aligned}
 \mathbb{D}[P \text{ INTERSECTION } Q](e) & \triangleq \mathbb{D}[P](e) \cap \mathbb{D}[Q](e) \\
 \mathbb{D}[P \text{ UNION } Q](e) & \triangleq \mathbb{D}[P](e) \cup \mathbb{D}[Q](e) \\
 \mathbb{D}[P \text{ MINUS } Q](e) & \triangleq \mathbb{D}[P](e) \setminus \mathbb{D}[Q](e)
 \end{aligned}$$

The second group of operators consists of arithmetic operators, and binary relations:

$$\begin{aligned}
 \mathbb{D}[f(P_1, \dots, P_n)](e) & \triangleq f(\mathbb{D}[P_1](e), \dots, \mathbb{D}[P_n](e)) \\
 \mathbb{D}[P \text{ R } X](e) & \triangleq \mathbb{D}[P](e) \text{ R } \mathbb{D}[Q](e)
 \end{aligned}$$

Any infix operator, such as $+$, $-$, $*$, $/$ will also be used in Elisa-D as an infix operation. For instance $45 + \text{Age}$ is a valid information descriptor. An example of the use of a binary relation is: $\text{Age} > 20$.

6.4.3.4 Type coercions

There exist some explicit forms of object type coercion in Elisa-D. These can be divided into two classes:

1. Conversion of the population of an information descriptor to a single value. This value can be used again as an information descriptor.
2. Conversion of the population of an information descriptor to a population of a different type.

These coercions are discussed successively. Coercions that lead to a single value of some label type typically perform some computation.

1. The function NUMBER OF counts the number of elements occurring in an information descriptor.

$$\mathbb{D}[\text{NUMBER OF } P](e) \triangleq \text{Cnt}(\mathbb{D}[P](e))$$

2. The function SUM adds the elements occurring in the first component of an information descriptor (including duplicates). This function is only applicable if addition is defined for the elements in the first component of the information descriptor involved.

$$\mathbb{D}[\text{SUM } P](e) \triangleq \text{Sum}(\mathbb{D}[P](e))$$

3. The functions MIN and MAX calculate the minimal and the maximal element occurring in the first component of an information descriptor. These functions require the existence of an ordering on the elements occurring in the first component of the involved information descriptor.

$$\begin{aligned} \mathbb{D}[\text{MIN } P](e) &\triangleq \text{Min}(\mathbb{D}[P](e)) \\ \mathbb{D}[\text{MAX } P](e) &\triangleq \text{Max}(\mathbb{D}[P](e)) \end{aligned}$$

For the second type of coercion the following operators are available:

1. Multiple occurrences are filtered from the result of an information descriptor by the use of the DISTINCT operator:

$$\mathbb{D}[\text{DISTINCT } P](e) \triangleq \text{ds}(\mathbb{D}[P](e))$$

2. The elements in an information descriptor P can be grouped into sets, according to a certain grouping criterion Q , using the group operator:

$$\mathbb{D}[\text{GROUP } P \text{ BY } Q](e) \triangleq \varphi(\mathbb{D}[P](e), \mathbb{D}[Q](e))$$

3. The coercion from sets to elements from these sets is achieved by the UNITE operator. Naturally, it is required that the elements in the first component of the involved information descriptor are sets.

$$\mathbb{D}[\text{UNITE } P](e) \triangleq \Upsilon(\mathbb{D}[P](e))$$

4. The elements in an information descriptor P can be ordered, according to an ordering criterion Q , using the sort operator:

$$\mathbb{D}[\text{SORT } P \text{ BY } Q](e) = \psi(\mathbb{D}[P](e), \mathbb{D}[Q](e))$$

Generators form two special operations of the second class, these are required for the formulation of special types of constraints.

$$\begin{aligned} \mathbb{D}[P \text{ TIMES } P'](e) &\triangleq \mathbb{D}[P](e) \diamond \mathbb{D}[P'](e) \\ \mathbb{D}[\text{ALL SUBSETS OF } P](e) &\triangleq \wp(\mathbb{D}[P](e)) \end{aligned}$$

As an example, the information descriptor President TIMES State pairs all presidents with all states, and ALL SUBSETS OF Ship yields all possible sets of ships (see figure 5.6 (page 83)). Obviously, all convoys are part of this information descriptor.

6.4.3.5 Miscellaneous operations

Information descriptors relate beginning and ending points of paths through the information structure. The THE operation, restricts an information descriptor to its beginning point. Selecting an arbitrary element from an information descriptor can be done by the A operation. These operations are defined by:

$$\begin{aligned} \mathbb{D}[\text{THE } P](e) &\triangleq f(\mathbb{D}[P](e)) \\ \mathbb{D}[\text{A } P](e) &\triangleq \alpha(\mathbb{D}[P](e)) \end{aligned}$$

Set comprehension can also be used in an information descriptor. This will, in general, not improve the readability of the resulting expressions, but with the aid of the (to be introduced) macro mechanism, these constructs may be made more readable. The set comprehension operation restricts the result of an information descriptor P to values adhering to a condition C :

$$\mathbb{D}[\{v \text{ IN } P \mid C\}](\langle M, V \rangle) \triangleq \{x \in \mathbb{D}[P](\langle M, V \rangle) \mid \mathbb{D}[C](\langle M, V \odot v : \{\langle x, x \rangle\} \rangle)\}$$

In information descriptors, the conditional clause is introduced as:

$$\mathbb{D}[\text{IF } C \text{ THEN } P \text{ ELSE } Q](e) \triangleq \text{if } \mathbb{D}[C](e) \text{ then } \mathbb{D}[P](e) \text{ else } \mathbb{D}[Q](e)$$

In Elisa-D, predicates are also treated as information descriptors. The basis for predicates is formed by the boolean values, which are introduced as special zero-adic operators:

$$\mathbb{D}[\text{TRUE}](e) \triangleq 1_{\mathcal{PE}} \quad \mathbb{D}[\text{FALSE}](e) \triangleq \emptyset_{\mathcal{PE}}$$

The Elisa-D pendants of the traditional operations from predicate calculus, such as \wedge , \vee will be introduced using the macro mechanism in the next subsection.

Finally, Elisa-D allows for the selection of instances of schema types as a restriction mechanism on the evaluation of information descriptors passing through schema types. This can be done by the **RESTRICT P TO SCHEMA s Q** construct. Let s be the name of a schema type, then the semantics of this construct is:

$$\mathbb{D}[\text{RESTRICT } P \text{ TO SCHEMA } s \text{ } Q](e) \triangleq \sigma(\mathbb{D}[P](e), x, \mathbb{D}[Q](e))$$

where $s = \text{ONm}(x)$. The following example illustrates the use of this expression, and the elegance of the disclosure schema:

```
RESTRICT
  President being Member of Party having as Member the President WITH Person name 'B. Clinton'
TO SCHEMA
  Amrh INVOLVED IN Amev OF Time 20-9-93
```

This expression will give all the presidents belonging to the same political party as president B. Clinton, according to the application model recording history valid at 20-9-93 (see also figure 4.11 (page 77)). Usually, one may want to restrict any query P to the most recent application model recording history, a concrete Elisa-D interpreter should do this.

6.4.4 Evolution dependency

Information descriptors are used to specify constraints, and in the next chapter they will be employed in the definition of activities.

The Depends relation, introduced in section 3.3, capturing evolution dependencies between elements from the application model, are defined for Elisa-D using a set of recursive derivation rules. Here, we only specify the atomic derivation rules concerning names from the lexicon. The other (recursive) rules can easily be derived, by closely following the syntax of the information descriptors.

$$\begin{aligned} \text{Lexicon}(n, P) \wedge x \in \text{Referenced}(P) &\vdash n \text{ Depends } x \\ c \in D \wedge D \in \mathcal{D} &\vdash \text{CNm}(c) \text{ Depends } D \end{aligned}$$

where $\text{Referenced}(P)$ returns the set of object types referenced to in path expression P . The Referenced function can also be defined recursively, in terms of the structure of path expressions, with as initial values:

$$\begin{aligned} \text{Referenced}(p) &\triangleq \{\text{Base}(p), \text{Fact}(p)\}, \text{ for any } p \in \mathcal{P} \\ \text{References}(x) &\triangleq \{x\}, \text{ for any } x \in \mathcal{O} \end{aligned}$$

6.5 Language Enrichment

In this subsection we introduce a macro mechanism. The purpose of this mechanism is to enrich the retrieval language, by assigning meaning to more constructs. Each macro can be seen as a new grammar rule, extending the retrieval language.

6.5.1 Defining and applying macros

The general format of a macro definition is as follows:

$$\underline{\text{LET}} \ \omega_0 \ X_1 \ \omega_1 \ \dots \ X_n \ \omega_n \ \underline{\text{BE}} \ E$$

where X_1, \dots, X_n are names of variables, and the string $\omega_0, \dots, \omega_n$ is the name of the macro. The macro definitions form the second class of `Assignments`. Due to the macro definition, the expression $\omega_0 \ P_1 \ \omega_1 \ \dots \ P_n \ \omega_n$ is meaningful, and evaluated according to the actual grammar, at each point in time. As stated before, we apply lazy evaluation for the evaluation of macros by the ϵ operation:

$$\begin{aligned} & \mathbb{D}[\omega_0 \ P_1 \ \omega_1 \ \dots \ P_n \ \omega_n] (\langle M, V \rangle) \triangleq \\ & \epsilon \left(\lambda \langle t, \text{Pop} \rangle. \bigcup_{M_t(\omega_0 \ X_1 \ \omega_1 \ \dots \ X_n \ \omega_n) \downarrow} \mu \cdot \mathbb{D}[M_t(\omega_0 \ X_1 \ \omega_1 \ \dots \ X_n \ \omega_n)[X_i := P_i]_{i=1}^n] (\langle M, V \rangle, \text{Pop}) \right) \end{aligned}$$

where $E[X_i := P_i]_{i=1}^n$ represents the substitution in E of X_i by P_i for all $1 \leq i \leq n$.

There are two ways to introduce a macro into the environment (M). A macro can be defined globally, for all points in time, or just for the current (evaluation) time:

$$\begin{aligned} & \mathbb{A}_t[\underline{\text{LET}} \ \omega_0 \ X_1 \ \omega_1 \ \dots \ X_n \ \omega_n \ \underline{\text{BE}} \ P] (\langle M, V \rangle) \triangleq \\ & \langle M \oslash \langle t, \omega_0 \ X_1 \ \omega_1 \ \dots \ X_n \ \omega_n \rangle : \mathbb{D}[P] (\langle M, V \rangle), V \rangle \\ & \mathbb{A}_t[\underline{\text{LET}} \ \omega_0 \ X_1 \ \omega_1 \ \dots \ X_n \ \omega_n \ \underline{\text{ALWAYS BE}} \ P] (\langle M, V \rangle) \triangleq \\ & \langle M \oslash_{s \in \mathcal{T}} \langle s, \omega_0 \ X_1 \ \omega_1 \ \dots \ X_n \ \omega_n \rangle : \mathbb{D}[P] (\langle M, V \rangle), V \rangle \end{aligned}$$

This mechanism may easily render a grammar ambiguous. A possible strategy is to accept ambiguity on the basis of the observation that natural language is also ambiguous. Another approach is the introduction of a validity check upon entering new grammar rules. A possible means of overcoming ambiguity is to introduce priorities for macros. In the remainder of this subsection, we provide some examples of the use of the macro mechanism.

6.5.2 Simple examples

Some simple, yet illustrative, examples of the use of the macro mechanism are:

LET P AND ALSO Q ALWAYS BE (THE P) INTERSECTION (THE Q)
LET P OR ELSE Q ALWAYS BE (THE P) UNION (THE Q)
LET P BUT NOT Q ALWAYS BE (THE P) MINUS (THE Q)

With the above definitions, variants of the normal set operations are introduced, who operate on the begin points of information descriptors (and their underlying path expression), and ignore their ending points. As an illustration of the advantage of such a restriction, consider the expression:

President being Member of Party WITH Party name 'Republican'
 AND ALSO
 has as Recreation the Hobby WITH Hobby name 'Model trains'

yields all presidents from the republican party, which have model trains as their hobby. On the other hand, the expression:

President being Member of Party WITH Party name 'Republican'
INTERSECTION
has as Recreation the Hobby WITH Hobby name 'Model trains'

is always empty, as the intersection of the ending points: 'Republican' and 'Model trains' are always empty.

Another simple example is the introduction of correlation:

LET P THAT O ALWAYS BE (P OF O) INTERSECTION O

To find the presidents who where inaugurated at an age younger than 45 years, i.e. inaugurated at least once within 45 years of *their* birth year, a convenient formulation is now:

President being the Head of the Administration having its Inauguration in Year WITH Year-nr
 \leq
45 + Year-nr IS NAME OF Year when was Born THAT President

This is called a correlation expression.

6.5.3 Connector names

One way to make Elisa-D sentences more readable, is the introduction of connector names (also referred to as role names). In the EVORM (and ER) schemas provided in previous chapters, we usually attributed predicates with connector names, rather than predicate names.

A connector name verbalises the transition from one object type to another through a fact type via a combination of a predicate and a reverse predicate. Some examples of the introduction of connector names are:

LET of birth of ALWAYS BE when was Born the
LET being the president of ALWAYS BE being the Head of the
LET contested in ALWAYS BE having Election results in

These connector names allow for such formulations as:

Year of birth of President
Person being the president of Administration
Person contested in Election

The notion of *connector names* can be formalised by introducing the function $\text{Connector} : \mathcal{P} \times \mathcal{P} \rightarrow \text{Names}$, such that:

$$\text{Connector}(p, q) = n \Rightarrow \text{Fact}(p) = \text{Fact}(q)$$

For every connector name n , such that $\text{Connector}(p, q) = n$, the following macro definition can be introduced:

LET n ALWAYS BE $\text{PNm}(p)$ $\text{ONm} \cdot \text{Fact}(p)$ $\text{RNm}(q)$

6.5.4 Instance denotations

Some constructions are introduced that facilitate the denotation (identification) of object instances used in information descriptors in an orthogonal way in this section. This denotation mechanism is defined using a set of mutually recursive macro definitions.

The following macro definition can be associated with any label type $l \in \mathcal{L}$:

$$\underline{\text{LET}} \text{ ONm}(l) : P_1 \underline{\text{ALWAYS BE}} \text{ ONm}(l) P_1$$

Before introducing the denotations of the other classes of object types, we define for any $X \subseteq \mathcal{O}$, its complete family as: $X_f \triangleq \{x \in \mathcal{O} \mid \exists y \in X [y \text{ RootOf } x]\}$.

Instances of entity types, without an underlying structure, are denoted in terms of other object type denotations, referred to as identification paths. These identification paths relate the entity type instances to instances of other object types via fact types. For this purpose, we presume the existence of a function: $\text{Ident} : \mathcal{E}_f \rightarrow (\mathcal{P} \times \mathcal{P})^+$. Some well formedness properties for Ident are discussed, in [Hof93]. Let $\text{Ident}(e) = \langle \langle p_1, q_1 \rangle, \dots, \langle p_n, q_n \rangle \rangle$, then the following denotation convention can be associated to e :

$$\begin{aligned} &\underline{\text{LET}} \text{ ONm}(e) : P_1, \dots, P_n \underline{\text{ALWAYS BE}} \\ &\quad \text{ONm}(e) \\ &\quad \text{PNm}(p_1) \text{ RNm}(q_1) \text{ ONm} \cdot \text{Base}(q_1) : P_1 \\ &\quad \text{AND ALSO} \\ &\quad \dots \\ &\quad \text{AND ALSO} \\ &\quad \text{PNm}(p_n) \text{ RNm}(q_n) \text{ ONm} \cdot \text{Base}(q_n) : P_n \end{aligned}$$

Two ways of denotation exist for fact types instances. If an order exists on the predicates of the fact type, then the instances can be denoted using this standard order. Otherwise, the predicate names should be used to denote the fact type instance.

Let $\text{Order} : \mathcal{F} \rightarrow \mathcal{P}^+$ be a function such that $f = \text{Set} \cdot \text{Lin} \cdot \text{Order}(f)$ for any $f \in \text{dom}(\text{Order})$. Then Order defines an order on the predicates in fact types. Let $f \in \mathcal{F}_f$, and $\text{Order}(f) = \langle P_1, \dots, P_n \rangle$, then instances of f can be denoted according to the following format:

$$\begin{aligned} &\underline{\text{LET}} \text{ ONm}(f) : P_1, \dots, P_n \underline{\text{ALWAYS BE}} \\ &\quad \text{ONm}(e) \\ &\quad \text{RNm}(p_1) \text{ ONm} \cdot \text{Base}(p_1) : P_1 \\ &\quad \text{AND ALSO} \\ &\quad \dots \\ &\quad \text{AND ALSO} \\ &\quad \text{RNm}(p_n) \text{ ONm} \cdot \text{Base}(p_n) : P_n \end{aligned}$$

Further, if $f \in \mathcal{F}_f$, and $f = \{p_1, \dots, p_n\}$, instances of f can (alternatively) be denoted according to:

$$\begin{aligned} &\underline{\text{LET}} \text{ ONm}(f) : \text{PNm}(p_1) : P_1, \dots, \text{PNm}(p_n) : P_n \underline{\text{ALWAYS BE}} \\ &\quad \text{ONm}(e) \\ &\quad \text{RNm}(p_1) \text{ ONm} \cdot \text{Base}(p_1) : P_1 \\ &\quad \text{AND ALSO} \\ &\quad \dots \\ &\quad \text{AND ALSO} \\ &\quad \text{RNm}(p_n) \text{ ONm} \cdot \text{Base}(p_n) : P_n \end{aligned}$$

The denotation of power type, sequence type, and schema types, is rather straightforward. Let $g \in \mathcal{G}_f$, $s \in \mathcal{S}_f$, and $c \in \mathcal{C}_f$, then:

$$\begin{aligned} &\underline{\text{LET}} \text{ ONm}(g) : \{P_1, \dots, P_n\} \underline{\text{ALWAYS BE}} \text{ ONm}(g) \{ \text{ONm} \cdot \text{Elt}(g) : P_1, \dots, \text{ONm} \cdot \text{Elt}(g) : P_n \} \\ &\underline{\text{LET}} \text{ ONm}(s) : \langle P_1, \dots, P_n \rangle \underline{\text{ALWAYS BE}} \text{ ONm}(g) \langle \text{ONm} \cdot \text{Elt}(g) : P_1, \dots, \text{ONm} \cdot \text{Elt}(g) : P_n \rangle \\ &\underline{\text{LET}} \text{ ONm}(c) : [X_1 \rightarrow \{P_1^1, \dots, P_{m_1}^1\}, \dots, X_n \rightarrow \{P_1^n, \dots, P_{m_n}^n\}] \underline{\text{ALWAYS BE}} \\ &\quad \text{ONm}(c) [X_1 \rightarrow \{X_1 : P_1^1, \dots, X_1 : P_{m_1}^1\}, \dots, X_n \rightarrow \{X_n : P_1^n, \dots, X_n : P_{m_n}^n\}] \end{aligned}$$

6.5.5 Predicates and constraints

Predicates can be introduced using the macro mechanism. The test whether an information descriptor has an empty result provides another illustration of the use of the conditional clause:

LET SOME P ALWAYS BE IF P THEN TRUE ELSE FALSE

As an example, the predicate SOME X x tests at each point in time, whether x is an instance of object type X .

Using the above operators, some further notational shorthands can be defined. The traditional operations on predicates can be formed in the usual fashion, using logical connectives and quantification:

LET C_1 AND C_2 ALWAYS BE (SOME C_1) INTERSECTION (SOME C_2)

LET C_1 OR C_2 ALWAYS BE (SOME C_1) UNION (SOME C_2)

LET NO C ALWAYS BE TRUE MINUS C

LET FOR SOME x IN P HOLDS C ALWAYS BE SOME $\{x \text{ IN } P \mid C\}$

LET FOR EACH x IN P HOLDS C ALWAYS BE NOT FOR SOME x IN P HOLDS NOT C

Set comparison operators can also be introduced:

LET P_1 INCLUDES P_2 ALWAYS BE NO (THE P_2) MINUS (THE P_1)

LET P_1 EQUALS P_2 ALWAYS BE (P_1 INCLUDES P_2) AND (P_2 INCLUDES P_1)

Predicates are used to formulate constraints in LISA-D, in particular those constraints which cannot be represented graphically. LISA-D has also been extended with textual versions of the graphical constraints used in PSM schemas. Elisa-D can be extended in a similar way with such constraints as:

UNIQUE($F_1 : p_1, \dots, F_n : p_n$)

where p_i denotes a the name of a predicator occurring in the fact type with name F_i .

The introduction of predicates (information descriptors) as constraints, can be done formally by defining the set of constraint definitions as: $\gamma \triangleq \text{Information Descriptor}$, with semantics:

$$H \llbracket d \rrbracket_t \triangleq \mu_t \cdot \mathbb{D} \llbracket d \rrbracket (e, \text{Pop}) \neq \emptyset$$

where d is an Information Descriptor, e is the default environment containing at least the standard macro definitions, and Pop is the population of the disclosure schema. We are now also in a position to provide a formal definition of the lsConstrSet predicate for constraints in Elisa-D:

$$\text{lsConstrSet}(C) \triangleq C \subseteq \gamma$$

Note that axiom AMV12 requires that any object type referred to in a $c \in C$, must be present in the current application model version X . The lsConstrSet(C) could be refined further by plausibility checks, excluding contradicting constraints (see e.g. [Hof93]).

6.5.6 Recursive macro definitions

The macro mechanism offers the possibility to introduce recursion. As a first example, consider the following macro definition:

LET Fac n ALWAYS BE IF n EQUALS 0 THEN 1 ELSE $n * \text{Fac } (n - 1)$

The operation of this mechanism is illustrated by the following proof of SOME (Fac 2 EQUALS 2):

$$\begin{aligned} \text{Fac } 2 &\equiv \{\text{NOT SOME } 2 \text{ EQUALS } 0\} \\ 2 * \text{Fac } 1 &\equiv \{\text{NOT SOME } 1 \text{ EQUALS } 0\} \\ 2 * 1 * \text{Fac } 0 &\equiv \{\text{SOME } 0 \text{ EQUALS } 0\} \\ 2 * 1 * 1 &\equiv \{\text{elaborate}\} \\ 2 & \end{aligned}$$

The transitive closure of an information descriptor is defined by the following definition:

LET ANY REPETITION OF P ALWAYS BE
 IF (DISTINCT P) EQUALS (DISTINCT $P P$) THEN
 DISTINCT P
 ELSE
 ANY REPETITION OF $P P$

With this macro definition, Elisa-D becomes as expressive as the Datalog language ([Ull89]). Note that the ϵ operation, and the lazy evaluation of macros, enable us to define such recursive macros without ending up in an infinite loop.

Example 6.5.1

Figure 5.10 gives an example of the use of the transitive closure. According to this schema, activities may have a decomposition, consisting of substates and subactivities. Subactivities may also have a decomposition. The relation between activities, and their corresponding subactivities, subsubactivities, etc., is captured by the following expression:

ANY REPETITION OF (Activity being decomposed into Activity graph COMPRISING Activity)

where being decomposed into is the connector name of the decomposition fact type. This information descriptor relates activities to the activities occurring in their direct or indirect decompositions. \square

6.6 Special Evolution Related Constructs

The language introduced so far, does not exceed the functionality of LISA-D as yet. A limited number of extensions is sufficient, to accommodate queries concerned with evolution. These extensions range from extensions to the lexicon up to macro definitions.

6.6.1 Basic constructions

The name NOW is introduced to obtain a grip on the point in time at which the query is evaluated:

Lexicon(NOW, $\lambda t. \{ \langle t, t \rangle \}$)

Time intervals, during which an expression exists (i.e., has a nonempty result), can be obtained by the following operator:

$\mathbb{D}[\text{VALIDITY OF } C](\langle M, V \rangle) \triangleq \nu(\mathbb{D}[P](\langle M, V \rangle))$

The following operator unites the result of an information descriptor over its existence.

$\mathbb{D}[\text{ALL } P \text{ EVER}](\langle M, V \rangle) \triangleq \tau(\mathbb{D}[P](\langle M, V \rangle))$

Another verbalisation of this operator is introduced via:

LET WHICHEVER P ALWAYS BE ALL P EVER

6.6.2 Extra macro definitions

The name AT is introduced in the lexicon of Elisa-D to relate, in a generic sense, histories to points in time. When using the AT keyword, we do not want to distinguish between elements from \mathcal{AME} and Ω . Therefore, we first define Element as a generic term for application model elements, and their (possible) relation with values (from Ω):

LET Element ALWAYS BE Ame UNION Value INVOLVED IN Instance typing

Intuitively, AT should have the following semantics:

1. It relates values from Ω to points in time from \mathcal{T} , via fact type `HasTypes` and object type `AME` (see figure 3.10 (page 55)). This connection allows for the handling of time stamping.
2. It relates application model elements to \mathcal{T} . This offers the possibility to query different versions of the application model.

This semantics is summarised in the following macro definition:

LET AT ALWAYS BE Element INVOLVED IN Ame evolution OF Time

The macro `WHEN` is the reverse of `AT`:

LET WHEN ALWAYS BE Time INVOLVED IN Ame evolution OF Element

Some examples of the usage of these constructions are:

1. THE Time WHEN X
This yields all points in time at which information descriptor X has a nonempty result.
2. THE Squadron AT NOW
This results in all squadrons existing at the point in time at which the information descriptor is evaluated.

The `AT` and `WHEN` names relate points in time to the generalised notion of `Element`. The fact type `Holds` relates time intervals and application model elements. To obtain better readable sentences, we also enrich `Element` with this relation:

LET HAVING LIVESPAN ALWAYS BE Element INVOLVED IN Holds OF Time interval

LET LIVESPAN OF ALWAYS BE Time interval INVOLVED IN Holds OF Element

The operations on \mathcal{T} and \mathcal{T}_I , introduced in subsection 6.3.2, are available for `Elements`, by defining for instance:

LET DURING ALWAYS BE HAVING LIVESPAN DURING

LET BEFORE ALWAYS BE (HAVING LIVESPAN UNION AT) BEFORE

LET AFTER ALWAYS BE (HAVING LIVESPAN UNION AT) AFTER

LET CONTAINING ALWAYS BE (HAVING LIVESPAN UNION AT) CONTAINS

LET INCREMENT OF ALWAYS BE AT IS INCREMENT OF

Note that `BEFORE`, `AFTER` and `CONTAINS` are defined for any kind of combination between time interval and point in time. Another abbreviation allowing for easy formulations of temporal selections is introduced as:

LET X [Y] Z ALWAYS BE X (DISTINCT THE Y) Z

This construction allows for sentences such as:

President being Member [DURING 1993] of Party 'Republican'

resulting in the presidents which were a member, during 1993, of the republican party. On the other hand, the sentence:

President [DURING 1993] being Member of Party 'Republican'

results in all US president, during 1993, which were members of the republican party.

6.6.3 Time modalities

A special class of time related operations is formed by the time modalities. We will demonstrate the expressiveness of Elisa-D by showing how such operators are introduced via the macro mechanism.

The first operator corresponds to the next (\square) operator in temporal logic. This operator is (verbosely) introduced by:

LET AT THE NEXT POINT OF TIME C WILL HOLD ALWAYS BE
SOME VALIDITY OF C CONTAINING INCREMENT OF NOW

Statements about system behaviour in the future can be formulated via the following constructs, from which the first corresponds to the ever (\diamond) operator from temporal logic:

LET AT SOME TIME C WILL HOLD ALWAYS BE SOME VALIDITY OF C AFTER NOW
LET NEVER C WILL HOLD ALWAYS BE NOT AT SOME TIME C WILL HOLD
LET FROM NOW ON C ALWAYS BE NOT SOMETIME NOW BEFORE VALIDITY OF NOT C

Properties from the past can be established by:

LET AT SOME TIME C HAS HELD ALWAYS BE SOME VALIDITY OF C BEFORE NOW
LET NEVER C HAS HELD ALWAYS BE NOT AT SOME TIME C HAS HELD

Properties over the entire history, including past and future, of the system can be addressed by:

LET ALWAYS C ALWAYS BE NOT SOME VALIDITY OF NOT C

6.7 An Example of Evolution

Finally, as an illustration of an evolving universe of discourse, we consider the workings of an insurance company for cars. The insured car and client are recorded for each policy sold. Every insured car has an associated registration number and type (Opel Corsa 1.2S, Ford Sierra 1.8, etc). A client is identified by name and address. The information structure of this universe of discourse is modelled in figure 6.10. As an illustration of Elisa-D's ability to be employed for the disclosure of (E)ER schemas, we use the (E)ER modelling technique for this example. Note that we have prefixed the attributes of an entity type with a #, and have associated predicator and predicator reverse names to predicators, in the format p/r .

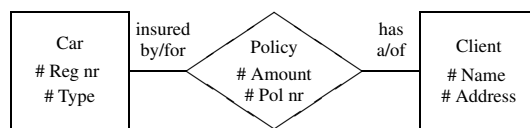


Figure 6.10: The information structure of a car insurance company

After some time, the insurance company notices a substantial difference between damage claims made for private cars, and for company cars. Rather than raising overall policy prices, price differentiation is used. Prices for new policies are increased by some percentage for company owned cars. Prices for new policies for private cars, however, are made dependent on car usage, measured in kilometers per year.

As these changes in price only involve new policies, the current population of the schema does not have to be altered. The evolved information structure is depicted in figure 6.11. The differentiation between private and company cars, leads to a subtyping of cars, and the dependency of the policy price on the amount of kilometers driven leads to the introduction of an extra entity type (Kilometrage) and relation type (Usage).

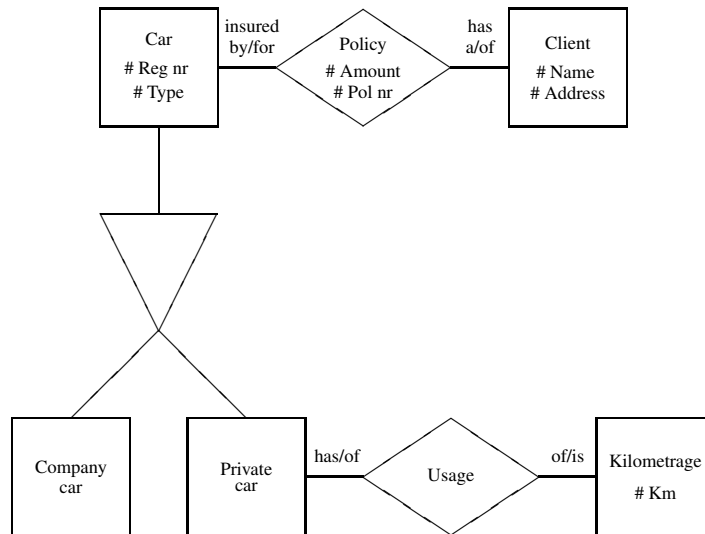


Figure 6.11: Car insurance with differentiated pricing

After this change, a large number of small companies not using their cars intensively, started to protest against new policy pricing, threatening to take their policies elsewhere. Thereupon, the insurance company also decided to differentiate policy pricing for company cars, as a result, subtyping cars into company cars and private cars was abolished. A further means to be more competitive, was found in the introduction of a reduction for clients not claiming damage. This reduction depends on the number of damage free years.

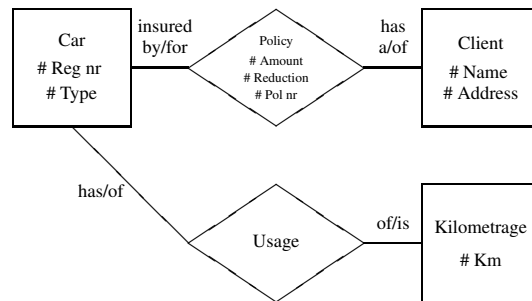


Figure 6.12: The final information structure

For the information structure, this leads to the introduction of the attribute *# Reduction* for relationship *Policy*. This results in figure 6.12. The introduction of the reduction, also requires a change in the current population of the information structure, as an initial reduction must be issued. Some illustrative examples of queries on instances, and the schema level, in the context of this example are:

1. Provide a list of all policies ever sold:

Policy

Note that the identification of policies changes over time, and therefore the standard names of its instances.

2. Provide a list of all policies during the year 1992:

Policy DURING Interval: 1992-01-01..1992-12-31

where we presume time intervals to be denoted as $d_1..d_2$.

3. Provide all fact types ever known about cars:

Fact type IN Types INVOLVED IN Instance typing OF Car

4. Provide all fact instances known about the cars registered during 1967-05-22:
Value OF Car DURING 1967-05-22
5. When was the concept of company car introduced:
STARTS VALIDITY OF Object type 'Company cars'
6. Which customers have left the company after the differentiation in private and company car (which happened at t_1), but have returned after the newest change (at t_2):
Client
WHICH EVER has a Policy AT t_1
AND-ALSO
WHICH EVER NOT has a Policy HAVING LIVESPAN CONTAINING $t_1..t_2$
AND ALSO
AT NOW
7. The number of policies sold for cars with a usage of 10.000 kilometers during 1993.
This sentence is highly ambiguous, and has at least the following two interpretations:
 - (a) The number of policies sold for cars with a usage during the year 1993 of 10.000 kilometers:
NUMBER OF Policy for Car having Usage [DURING 1993] of Kilometrage 10.000
 - (b) the number of policies sold during the year 1993 for cars with a usage of 10.000 kilometers:
NUMBER OF Policy [DURING 1993] for Car having Usage of Kilometrage 10.000

This example finalises this chapter.

Chapter 7

Manipulation of Evolving Application Models

He was wrong in thinking that the atmosphere of the Earth had closed finally and for ever above his head.

He was wrong in thinking that it would ever be possible to put behind the tangled web of irresolutions into which his galactic travels had dragged him.

He was wrong to think he could now forget that the big, hard, oily, dirty, rain-bow hung Earth on which he lived was a microscopic dot on a microscopic dot lost in the unimaginable infinity of the Universe.

From: "So Long, and Thanks for all the Fish",
Douglas Adams, Pan Books Ltd.

7.1 Introduction

In the previous two chapters, the evolution of the world model part of the application model, was elaborated for a concrete query language and modelling technique. In this chapter, the elaboration of the application model evolution is completed by the application of the general evolution theory to a modelling technique for the action model. The action modelling technique under consideration is Hydra. The resulting technique, supporting evolution, is referred to as Hydrae. A motivation for Hydra has been provided in section 1.4.

Hydra's semantics has been defined in [Hof93], in terms of Process Algebra [BW90a]. Its diagrams are employed to capture the dynamic aspects of a universe of discourse, together with their interaction with the static aspects. Hydra, originates from the meta modelling domain, and uses the concept of task as a main notion in the modelling of activities. Hydra diagrams model the initiation of tasks, moments of *choice*, *parallelism*, *synchronisation*, and *updates* of populations and intermediate results.

A short introduction to the syntax and semantics of Hydrae diagrams is provided in the next section. This is followed, in section 7.3, by the formal definition of the syntax of Hydrae. The formal semantics of Hydrae is the same as Hydra, and will therefore not be discussed in this chapter. Activities (methods) modelled as a Hydra diagram are decomposable into sub-activities. The most elementary ones, correspond to elementary updates, in some appropriate language. LISA-D was used for this purpose in [Hof93], Elisa-D will be used in section 7.4.

7.2 Informal Introduction to Hydrae

Prior to defining the syntax of Hydrae formally, we provide a short introduction to its history, and the underlying concepts. Task structures were introduced in [Bot89] as a modelling technique for problem solving processes. The graphical formalism of task structures, and the possibility to decompose tasks, offers an easy way to view tasks at a different level of abstraction. A proper semantics of task structures is defined in [HN93], that covers sequential execution, iteration, choice, parallelism, and synchronisation. Task structures deal solely with dynamic aspects, and lack a formal relationship with any underlying data model. Hydra ([Hof93]), extended task structures with data modelling aspects, and integrated task structures with the PSM data modelling technique, and LISA-D. Hydrae, will extend Hydra with the notion of evolution. Due to its origins in problem solving processes, Hydrae uses the concept of task as its main modelling concept. This notion, however, directly corresponds to the notions of activity and method specification used in earlier chapters. In this chapter, we follow the Hydra tradition, and use the notion of task.

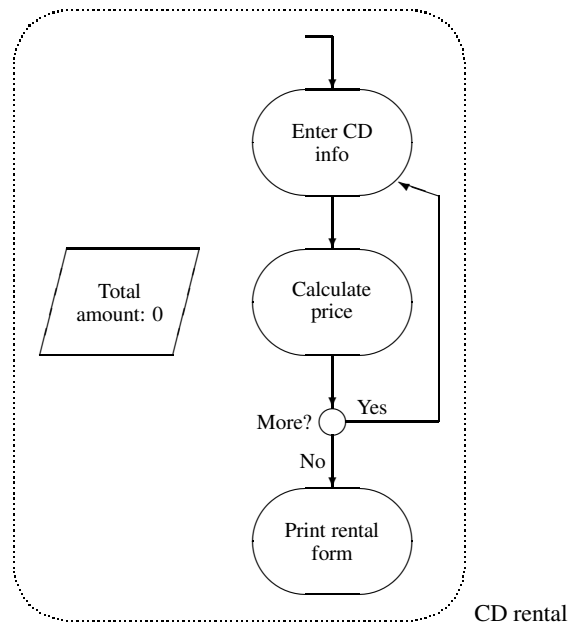


Figure 7.1: Rental of CDs

In Hydrae, a task may be defined in terms of other tasks, its *subtasks*. Performing a task, may involve choosing between different subtasks. Therefore, *decisions* are used to make such choices explicit. During the execution of a task, a subtask (or decisions) may trigger other subtasks by *triggers*. A sample Hydra diagram for the rental of CDs is provided in figure 7.1 to illustrate these concepts. The task CD rental consists of three subtasks:

Enter CD information, Calculate rental price, and Print rental form,
and one decision point:

More CDs?.

The task Enter CD information is designated as the initial task of the composed task CD rental. When the execution of CD rental reaches the More CDs decision point, the associated decision rules (Elisa-D information descriptor) are evaluated, and the execution of the task is continued with the chosen subtask(s).

Tasks may use and change the population present in the application model. As well as this global and relatively stable information, information of a more local and temporary nature exists. Due to its temporary nature, this latter information is not part of the population, to accommodate such information, local variables are introduced. The internal variable Total amount, depicted in figure 7.1, is used by the subtasks to calculate the total amount of the rental fee. A local variable may receive an initial value. (In this case 0.)

When executing a task with subtasks, the execution path may split-up. In figure 7.2, which is borrowed from [Hof93],

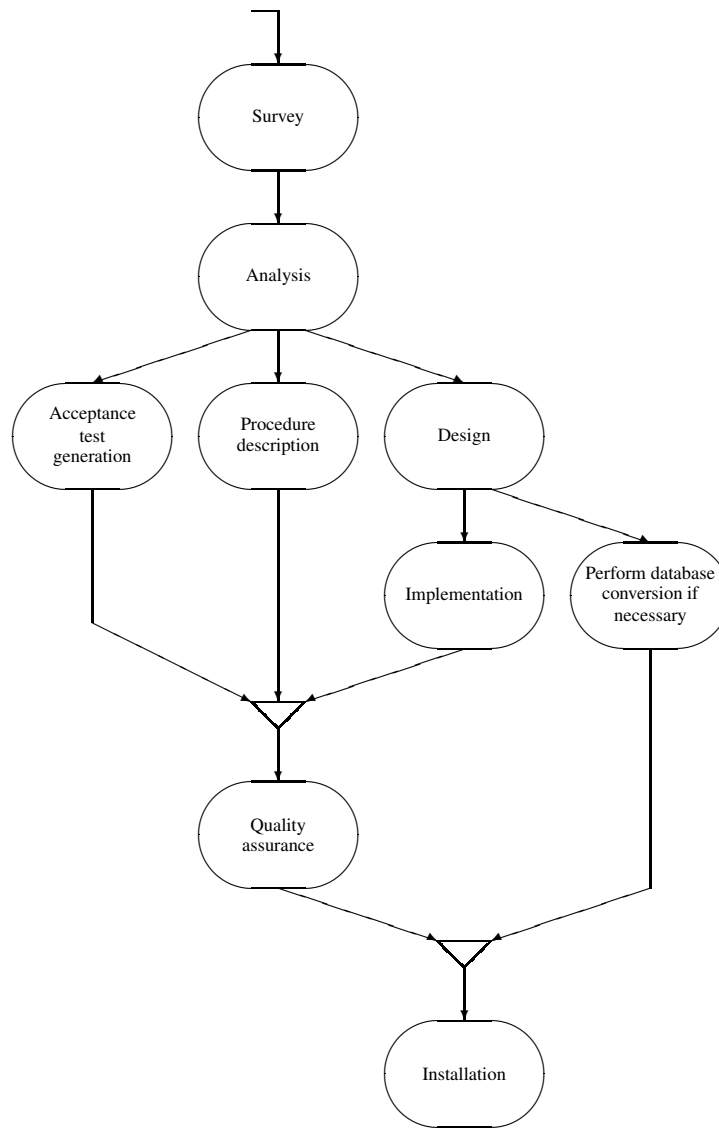


Figure 7.2: Yourdon way of working

an example of such a split-up is provided. When the Analysis task is finished, the execution path is split-up into three, and eventually four separate execution paths. In some situations, one may want to synchronise these execution paths before commencing a next task. For instance, all preparations must be finished before the Installation of the new system. To this end, the synchronisation primitive of Hydra can be employed. This primitive is used to synchronise two or more execution paths in a task. The subtask Quality assurance will not start until Acceptance test generation, Procedure description, and Implementation have finished.

During the execution of a task, several decision points may be encountered, terminating decisions are a special class of decision points. In general, a decision will always trigger another task(s). Sometimes, however, no next task can be triggered, as none of the associated decision rules leads to a positive result. These decisions are identified explicitly, since the formal definition of their semantics requires this, and it also supports plausibility checks on task structures. An example of a task with a terminating decision is provided in figure 7.3.

We have already introduced local variables as a mechanism to store data locally, as well as the local variables, (FIFO) buffers, can be used to store data locally. An example of the use of a buffer is provided in figure 7.4 ([Hof93]). The

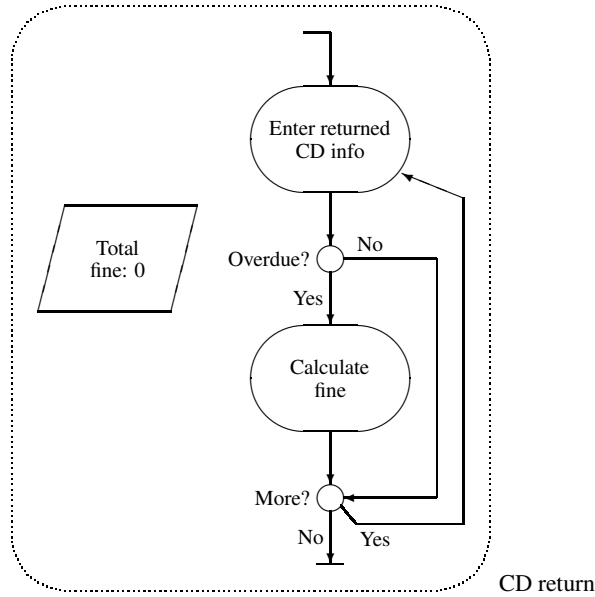


Figure 7.3: Return of a CD

Produce task produces items, and queues them in the Product FIFO queue, the Consume task consumes them.

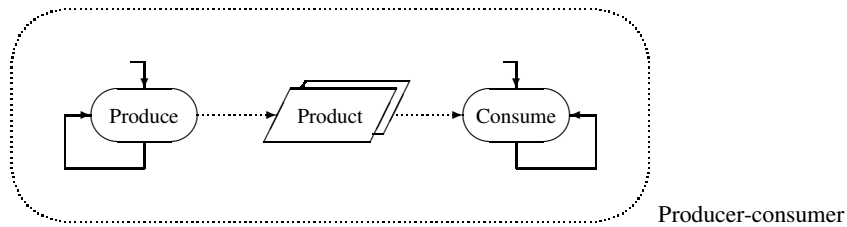


Figure 7.4: Example buffer

Any change of the application model brought about by the execution of a task should adhere to the constraints provided in the world model, as well as these global constraints, Hydrae also allows for the specification of *pre*- and *post-conditions* for tasks which can be used to enforce constraints on intermediate results. Those tasks which are elementary with respect to their execution, are referred to as transaction tasks. Transaction tasks may, however, still be decomposed into sub-tasks. Nevertheless, their execution is elementary, i.e. their execution may not be interrupted by other tasks outside the transaction task. An example of such a decomposable transaction task is provided in figure 7.5 ([Hof93]). Non decomposable transactions tasks are referred to as atomic. The semantics of such atomic transactions is provided in [Hof93] by extending LISA-D with elementary updates. The semantics of atomic transactions is defined by extending Elisa-D with elementary updates in section 7.4. As stated before, the semantics of Hydrae tasks is the same as in Hydra. Therefore, we do not elaborate on the latter semantics.

7.3 Syntax of Hydrae

In this section, the syntax of Hydrae is introduced formally. We start by defining a universe for Hydrae task structures. This universe limits the evolution space for the evolution of Hydrae action models.

7.3.1 Hydrae universe

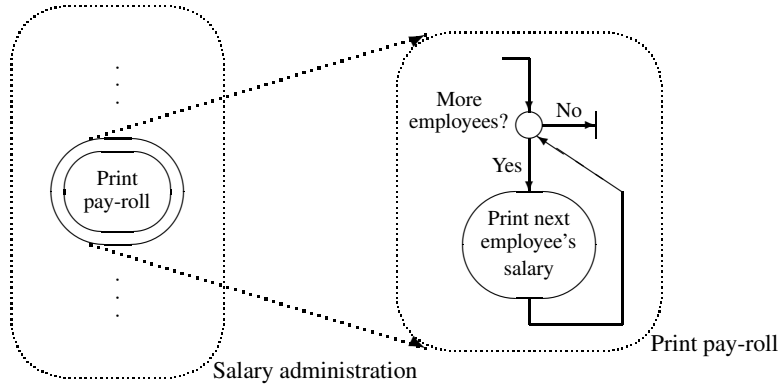


Figure 7.5: Example transaction task

The Hydrae universe is spanned by the following components:

1. A set \mathcal{X} of *task objects*.

\mathcal{X} itself, is the (disjoint) union of:

- a set of *synchronisers* \mathcal{W} ,
- a set of tasks μ (these tasks form the link to the general theory),
- and a set of *decisions* \mathcal{K} .

In \mathcal{K} , a subset \mathcal{K}^\perp is distinguished consisting of the *terminating decisions*, and in μ a subset \mathcal{J} is distinguished consisting of the *transaction tasks*.

2. A relation $\text{Trig} \subseteq \mathcal{X} \times (\mathcal{X} \cup \{\neg\})$ of triggers.

The element \neg is used as a special task object, defining a *terminating transition*.

3. A function $\text{Name} : \mu \rightarrow \text{Names}$ providing the name of a task.
4. Task structures are allowed to be decomposed. For this purpose, we introduce explicitly the notion of (task) body, conform procedure body.

Let β be the set of *task bodies*, then the function $\text{Body} : \beta \rightarrow \wp(\mathcal{X})$ provides the set of task objects contained in the decomposition of the body. The decomposition of a composed task is then associated by the partial function: $\text{CallOf} : \mu \rightarrow \beta$. The (indirect) decomposition mechanism allows composed tasks to share common subtasks. Consider for instance figure 7.6. There the sub-task A is used twice in the depicted composed task.

The decomposition hierarchy on tasks is defined by the relation:

$$d \text{ Sup } s \iff d \in \text{Body} \cdot \text{CallOf}(s)$$

If $d \text{ Sup } s$, this means that task object d is part of the decomposition of s .

5. A function $\text{Init} : \beta \rightarrow \mathcal{X}$ yielding the initial items of a task.
6. A function $\text{Buff} : \beta \rightarrow \wp(\text{Var})$, assigning buffers to the names of decomposed tasks.
The functions $\text{Cons}, \text{Prod} : \mu \rightarrow \wp(\text{Var})$, capture the consumption and production relation for buffers.
7. A function $\text{Locvar} : \beta \rightarrow \wp(\text{Var})$, providing the local variables.

8. An initialisation function $\text{Locinit} : \beta \times \text{Var} \rightarrow \text{Information Descriptor}$, assigning an initial value to a local variable.
9. Tasks may have associated pre- and post-conditions. These conditions are provided by the functions: $\text{Pre}, \text{Post} : \mu \rightarrow \text{Information Descriptor}$. Note that **TRUE** is a valid (dummy) condition.

The assigned information descriptor is interpreted as a predicate by evaluating it, and interpreting a nonempty result as *true*, and an empty result as *false*, thus **TRUE** always yields *true*.

10. A function $\text{Trans} : (\mu - \mu_d) \rightarrow \text{Transaction}$, providing an Elisa-D transaction for each atomic task (tasks which do not have an associated body), where $\mu_d \triangleq \text{dom}(\text{CallOf})$.

The syntactic category **Transaction** is defined in section 7.4.

11. Finally, decision rules for decision points, are associated to their outgoing triggers by the function:

$$\text{Choice} : \text{Trig}_{\mathcal{K}} \rightarrow \text{Information Descriptor}$$

where $\text{Trig}_{\mathcal{K}} \triangleq \{\langle x, y \rangle \in \text{Trig} \mid x \in \mathcal{K}\}$ is the set of triggers starting from decision points.

For instance, $\text{Choice}(k, x) = d$ is interpreted as: if d leads to a nonempty result, then x is triggered. $\text{Choice}(k, \rightarrow) = d$ is interpreted as: if d leads to a nonempty result, then the supertask may terminate (if no other subtasks are still being executed).

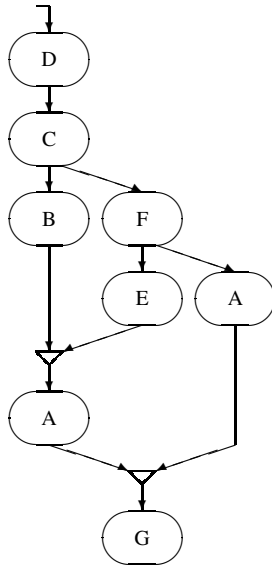


Figure 7.6: Multiple calls to a sub-task

The universe for Hydrae action models adheres to some rules that further refine this universe. As a start, terminating triggers can only be attached to terminating decisions, and terminating decisions must be able to terminate:

$$[\text{HU1}] \text{ (termination) } x \text{ Trig } \rightarrow \iff x \in \mathcal{K}^\perp$$

Triggers should not cross task boundaries:

$$[\text{HU2}] \text{ (trigger confinement) } x_1 \text{ Trig } x_2 \Rightarrow \exists_s [x_1 \text{ Sup } s \wedge x_2 \text{ Sup } s]$$

The initial item of a body, must be part of the decomposition associated to the body:

[HU3] (*initial item*) $b \in \beta \Rightarrow \text{Init}(b) \in \text{Body}(b)$

Transaction tasks, cannot be atomic, otherwise they would have an associated Elisa-D transaction.

[HU4] (*non-atomic transaction tasks*) $\mathcal{T} \subseteq \mu_d$

Tasks are only allowed to consume from, and produce for, buffers which are in the same decomposition as the task itself:

[HU5] (*locality*) $d \text{Sup } s \Rightarrow \text{Cons}(d) \cup \text{Prod}(d) \subseteq \text{Buff} \cdot \text{CallOf}(s)$

To avoid naming conflicts between local variables, buffers, and tasks, names are required to be disjoint:

[HU6] (*exclusive names*) The sets $\text{Locvar}(x)$, $\text{Buff}(x)$ and $\text{DecNames}(x)$ are disjoint for all tasks x , where:
 $\text{DecNames}(s) \triangleq \{\text{Name}(d) \mid d \text{Sup } s\}$.

Local variables may only receive an initial value for a task, if they are a local variable of the task given:

[HU7] (*initialisation*) $\text{Locinit}(x, v) \downarrow \Rightarrow v \in \text{Locvar}(x)$

This completes the universe of Hydrae models. The next step is to define the well-formedness axioms for versions of Hydrae models.

7.3.2 Hydrae versions

A version of the action model is identified, in the general evolution theory, by \mathcal{M}_t (see section 3.4), from this we derive the active set of tasks associated to that version:

$$\mu_t \triangleq \pi_2(\mathcal{M}_t)$$

Versions of Hydrae models, are required to have a unique task in the top of the decomposition hierarchy, the decomposition function should therefore be undefined for this task, i.e.:

[HV1] (*unique top*) $\exists! d \in \mu_t [\neg \exists s \in \mu_t [d \text{Sup } s]]$

The Sup relation can be generalised to a relation on \mathcal{M} as follows:

$$\langle x, d \rangle \text{Sup}_{\mathcal{M}} \langle x, s \rangle \triangleq d \text{Sup } s$$

Using this relation, we can define the following rule demanding the completeness of a Hydrae version with respect to decomposition.

[HV2] (*temporal conformity*) $s \in \mathcal{M}_t \wedge d \text{Sup}_{\mathcal{M}} s \Rightarrow d \in \mathcal{M}_t$

A direct result, is the following corollary stating that the active set of task definitions associated to a version, is complete with respect to the decomposition of tasks:

Corollary 7.3.1 $s \in \mu_t \wedge d \text{Sup } s \Rightarrow d \in \mu_t$

This completes the set of well-formedness rules on versions of Hydrae models. The IsActMod predicate that designates which action model versions are correct, can now be defined as:

Definition 7.3.1

$$\text{IsActMod}(\mathcal{M}_t) \triangleq \mathcal{M}_t \text{ adheres to the HV axioms}$$

□

The meta model of Hydrae is provided in figure 7.7. This meta model provides yet a further refinement of the disclosure schema defined in the previous chapter, and as a result, the Hydrae task structures can also be queried.

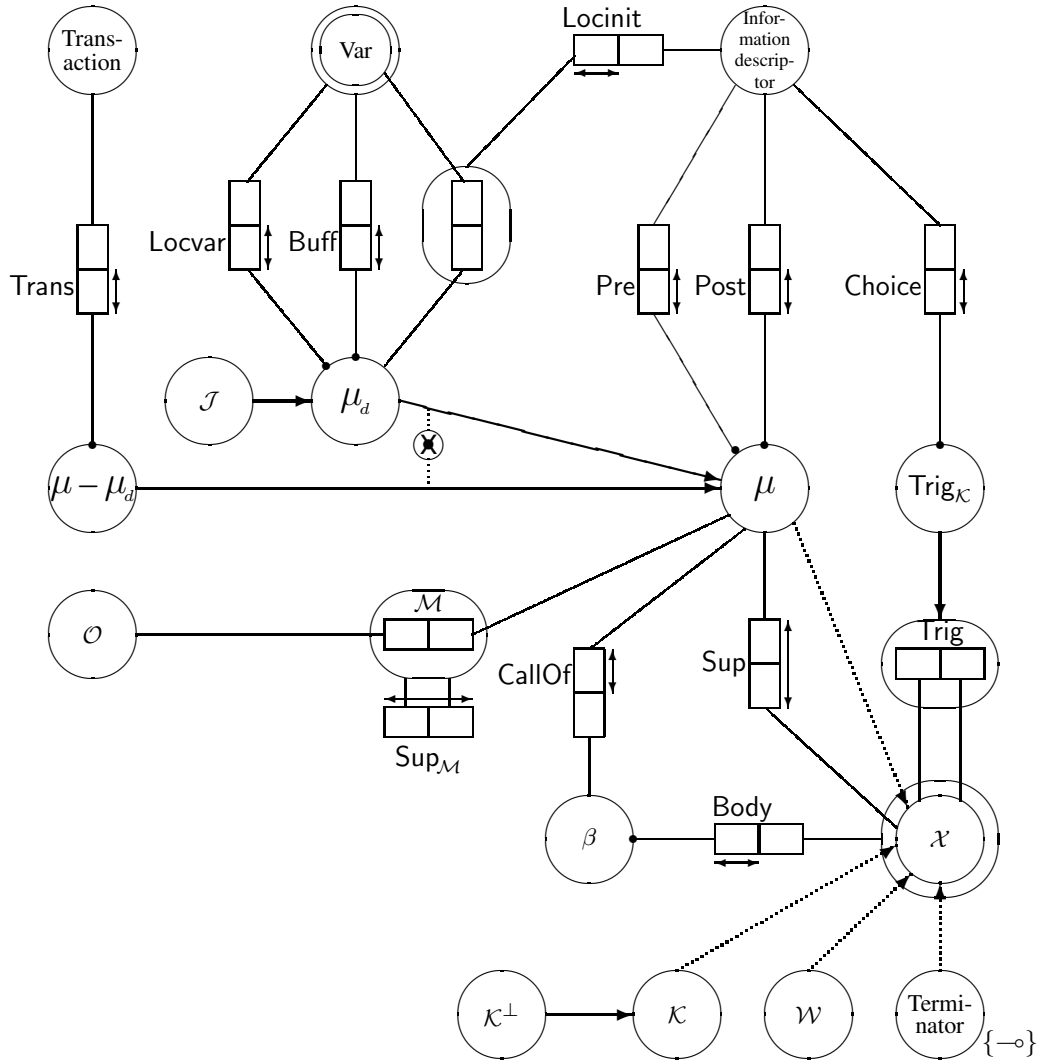


Figure 7.7: Hydrae Meta Schema

7.3.3 Evolution well-formedness

For Hydrae the IsEvol predicate can be further refined. Again, this is a matter of taste. The effect of axiom EEW1 (page 99) can easily be expanded to cover evolution of Hydrae models also, by adding Sup to the PartOf relation, resulting in:

$$x \text{ PartOf } y \triangleq x = \text{Elt}(y) \vee y \prec x \vee \exists_p [\text{Base}(p) = x \wedge \text{Fact}(p) = y] \vee x \text{ Sup } y$$

7.3.4 Evolution dependency

The Depends relation, providing the evolution dependencies between the elements in an application model, can be further refined for tasks by the following derivation rules:

$$\begin{array}{l} \text{Locinit}(\text{CallOf}(a), v) \text{ Depends } x \vdash a \text{ Depends } x \\ \text{Pre}(a) \text{ Depends } x \vdash a \text{ Depends } x \\ \text{Post}(a) \text{ Depends } x \vdash a \text{ Depends } x \end{array}$$

$$\begin{array}{l}
\text{Trans}(a) \text{ Depends } x \vdash a \text{ Depends } x \\
\text{Choice}(\langle a, b \rangle) \text{ Depends } x \vdash a \text{ Depends } x \\
d \text{ Depends } x \wedge d \text{ Sup } s \vdash s \text{ Depends } x
\end{array}$$

The Depends relation for information descriptors is provided in the previous chapter, including the base rules for this (recursively defined) relation; it will be defined for transactions in the next section.

7.4 Elisa-D Transactions

A connection is made between Hydrae and Elisa-D in this section. In Hydrae, any atomic task has associated a transaction statement from Elisa-D. This section is concerned with the extension of Elisa-D with these transactions.

We start by introducing Elisa-D updates, these updates operate on the application model history, and either terminate an element evolution, change an element evolution, or add a new element evolution. Next, queries on application models are taken into consideration, in transactions, queries have to be evaluated and communicated to the environment of the information system. Finally, a formal definition of the Elisa-D transactions is provided.

7.4.1 Updates

The first class of transactions we consider are *updates*. The semantics of Hydrae updates is defined by a function:

$$\mathbb{U} : \mathcal{T} \times \text{Update} \times \mathcal{AMH} \times \text{ENV} \rightarrow \mathcal{AMH}$$

which evaluates an update, at a given point in time, in the context of an existing application model history and environment, and returns an updated application model history.

The semantics of updates of application model histories is expressed in terms of updates of versions of the population of the disclosure schema. These latter semantics are provided by the function:

$$\mathbb{U}_\infty : \text{Update} \times \text{POP}_\infty \times \text{ENV} \rightarrow \text{POP}_\infty$$

where POP_∞ is the set of possible versions of populations of the disclosure schema:

$$\text{POP}_\infty \triangleq (\mathcal{O} \cup \mathcal{O}^M) \rightarrow \wp(\Omega \cup \mathcal{AME})$$

The advantage of this approach is that this latter function corresponds to the semantics of LISA-D updates of snapshot populations, defined in [Hof93] and [HPW93]. As a result, we restrict ourselves to the ‘historical’ implications of updates.

A new syntactic category is introduced for update, *Constant Denotations*. The expressions in this category correspond to the constant denotations which can be used in information descriptors. The semantics of constant denotations is defined using macros in subsection 6.5.4. The *Constant Denotation* is identified by:

c	constants
v	variables
d_1, \dots, d_n	denotation of entities and facts
$p_1 : d_1, \dots, p_n : d_n$	alternative denotation of facts
$\{d_1, \dots, d_n\}$	denotation of power instances
$\langle d_1, \dots, d_n \rangle$	denotation of sequence instances
$[X_1 \rightarrow \{d_1^1, \dots, d_{k_1}^1\}, \dots, X_n \rightarrow \{d_1^n, \dots, d_{k_n}^n\}]$	denotation of schema instances

The semantics for the three classes of updates identified in section 4.1 are provided in the remainder of this subsection, and correspond to add, delete and change transactions.

7.4.1.1 Adding instances

Adding instances to a population is performed by the add statement. This statement has the format: $\text{ADD ONm}(x): d$. The effect of adding instances to a version of the population of the disclosure schema is identified by:

$$\mathbb{U}_\infty[\text{ADD ONm}(x): d](p_1, e)$$

The meaning of this statement is to enforce an extension of the current population, that populates information descriptor $\text{ADD ONm}(x): d$. More exactly, a minimal extension p_2 of the current population p_1 , such that information descriptor $\text{ONm}(x): d$ has no empty result in the extended population p_2 . The notion of minimal extension is captured as:

$$p_1 \sqsubseteq p_2 \triangleq \forall_{x \in \mathcal{O}} [p_1(x) \subseteq p_2(x)]$$

The meaning of the above add statement, is therefore formally expressed by:

$\mathbb{U}_\infty[\text{ADD ONm}(x): d](p_1, e)$ is a minimal population p_2 such that:

1. $\text{IsPop}(\mathcal{O}_t \cup \mathcal{O}^M, p_2)$,
2. $p_1 \sqsubseteq p_2$ and
3. $\mu_t[\mathbb{D}[\text{ONm}(x): d](e)](\lambda_{t \in \mathcal{T}} p_2) \neq \emptyset$

See [Hof93] for a more detailed definition of the add statement on snapshot populations. Although it is required for the new population p_2 to be such that $\text{IsPop}(\mathcal{O}_t \cup \mathcal{O}^M, p_2)$, it may be necessary to weaken this in a concrete case and only require IsPop to hold for a new population resulting after a complete transaction (see subsection 7.4.3). For a more elaborate discussion of the enforcement of constraints during and after transactions, see e.g. [EGH⁺92].

When adding instances, the application model history should also be updated appropriately. In this respect, three cases can be distinguished:

1. some element evolutions are not influenced by the update,
2. some are modified directly or indirectly (adding an instance may, for example, change the result of a subtype defining rule) by the update,
3. and some new element evolutions are created.

There is a difference in the treatment of changes of instances of object types from the application model and those from the meta model, therefore the semantics of an add is identified by:

$$\mathbb{U}_t[\text{ADD ONm}(x): d](H, e) \triangleq H'$$

where H' is the union of:

1. The unaffected evolutions:

$$\{h \in H \mid h \uparrow t\}$$
2. Changed element evolutions from the application model:

$$\{h[\langle v, X \rangle : X \neq \emptyset]_t \mid h \in H_{pop} \wedge \langle v, X \rangle = \text{Typing}(h(t), p_2)\}$$

Since the result of subtype defining rules may have changed due to the addition, the set of types associated to a value may also have changed, thus the set of types associated to values (v) have to be re-evaluated ($\text{Typing}(h(t), p_2)$).

Note that this does not apply to instances of the meta model.

3. New element evolutions from the application model:

$$\{\lambda_{s>t}. \text{Typing}(\langle v, \{x\} \rangle, p_2) \mid x \in \mathcal{O} \wedge \text{IsRoot}(x) \wedge v \in p_2(x) - p_1(x)\}$$

The $\text{IsRoot}(x)$ guarantees that this is a new instance, and not an old one which has become an instance of a subtype. Note that p_1 and p_2 are defined below, as the old and new population of the disclosure schema.

4. New element evolutions from the meta model:

$$\{\lambda_{s>t}. v \mid x \in \mathcal{O}^M \wedge v \in p_2(x) - p_1(x)\}$$

and further, where:

1. p_2 is defined as $\mathbb{U}_\infty[\text{ADD ONm}(x): d](p_1, e)$, and p_1 is the population of the disclosure schema of H at t ,
2. $h[x : C]_t$ is used as an abbreviation for:

$$\lambda s. \text{if } s \leq t \text{ then } h(t) \text{ else (if } C \text{ then } x \text{ else } \perp \text{ fi) fi}$$

3. and $\text{Typing}(\langle v, Y \rangle, p)$ is identified by:

$$\langle v, \{x \mid v \in p(x) \wedge \exists a \in Y [x \text{ ldfBy } a] \} \rangle$$

7.4.1.2 Deleting instances

The second class of updates under consideration are deletions. Instances can be deleted from a population by the delete statement, with the format $\text{DELETE ONm}(x): d$. Analogous to the add statement, the semantics of the delete statement is expressed in terms of the delete operation of the disclosure schema:

$$\mathbb{U}_\infty[\text{DELETE ONm}(x): d](p_1, e)$$

The meaning of this statement is the enforcement of a minimal reduction of the current population p_1 , that unpopulates $\text{ONm}(x): d$ (at t), i.e. a maximal part p_2 of the population p_1 , such that information descriptor $\text{ONm}(x): d$ has an empty result in the reduced population p_2 . Formally, this is expressed by:

$\mathbb{U}_\infty[\text{ONm}(x): d](p_1, e)$ is a maximal population p_2 such that:

1. $\text{IsPop}(\mathcal{O}_t \cup \mathcal{O}^M, p_2)$,
2. $p_2 \sqsubseteq p_1$ and
3. $\mu_t[\mathbb{D}[\text{ONm}(x): d](e)](\lambda_{t \in \mathcal{T}}. p_2) = \emptyset$

When deleting instances, the application model history must be updated accordingly, and when doing so two main cases can be distinguished. Some element evolutions are not influenced by the update, while some are influenced by the update. Again there is a difference in the treatment of changes of instances of object types from the application model, and those from the meta model. The semantics of a delete is therefore determined as:

$$\mathbb{U}_t[\text{DELETE ONm}(x): d](H, e) \triangleq H'$$

where H' is the union of:

1. The unaffected evolutions:

$$\{h \in H \mid h \uparrow t\}$$

2. Changed element evolutions from the application model:

$$\{h[\langle v, X \rangle : X \neq \emptyset]_t \mid h \in H_{pop} \wedge \langle v, X \rangle = \text{Typing}(h(t), p_2)\}$$

This set deals both with changes to the set of types associated to values, due to changed results of subtype defining rules, and terminated instances (an instance was defined as the association between a value, and a set of types).

- Deleted element evolutions from the meta model:

$$\{h[h(t) : h(t) \in p_2(x)]_t \mid h \in H - H_{type} \wedge h \downarrow t \wedge x \in \mathcal{O}^M\}$$

and further, where:

- p_2 is identified by $\mathbb{U}_\infty[\text{DELETE ONm}(x): d](p_1, e)$, and p_1 is the population of the disclosure schema of H at t ,
- and Typing and $h[x : C]_t$ are defined as above.

7.4.1.3 Changing instances

The final class of updates are changes. Elements from the application model may be changed to other elements. The change operation has the following format:

$$\mathbb{U}_t[\text{CHANGE ONm}(x_1): d_1 \text{ TO ONm}(x_2): d_2](e, H) \triangleq H'$$

Its semantics is expressed by the combination of a delete and add operation on the disclosure schema as:

$$p_2 \triangleq \mathbb{U}_\infty[\text{ADD ONm}(x_1): d_1](\mathbb{U}_\infty[\text{DELETE ONm}(x_2): d_2](p_1, e), e)$$

To identify uniquely the changed instances of the disclosure schema, the following two requirements must hold on the specified change:

- The involved object types are either both from the application model, or they are the same meta object type: $x_1, x_2 \in \mathcal{O} \vee x_1 = x_2$.

The motivation of this requirement lies in the strict separation between element evolutions for instances, object types, methods, etc. (see section 3.5).

- The change may only involve exactly one instance from x_1 and x_2 .

Therefore, $|p_1(x_1) - p_2(x_1)| = 1$ and $|p_2(x_2) - p_1(x_2)| = 1$. As a result of this requirement, the changed element evolutions can be uniquely identified.

In the definition of the change operation, the semantics of the add and delete operations are used for the element evolutions not involved in the change (not x_1 or x_2). For these evolutions, the semantics of the add and delete operations are used to deal with any indirectly changed instances. The semantics of a change is now identified by:

$$\mathbb{U}_t[\text{CHANGE ONm}(x_1): d_1 \text{ TO ONm}(x_2): d_2](e, H) \triangleq H'$$

where H' is the union of:

- The set of evolutions not directly influenced by the change:

$$\{h \in \mathbb{U}_t[\text{ADD ONm}(x_1): d_1](\mathbb{U}_t[\text{DELETE ONm}(x_2): d_2](H, e), e) \mid x_1 \not\leq h(t) \wedge x_2 \not\leq h(\triangleright t)\}$$

where $x \not\leq h(t) \triangleq (h(t) \in \mathcal{O}^M \wedge h(t) = x) \vee \exists_{v, X} [h(t) = \langle v, X \rangle \wedge x \in X]$, captures the intuition: x is not involved in evolution h at t .

- Changed element evolutions, directly influenced by the change:

$$\{h[\text{Typing}(\langle v, \{x_2\} \rangle, p_2)]_t \mid h \in H_{pop} \wedge h(t) \leq_{x_1} p_1(x_1) - p_2(x_1) \wedge v \in p_2(x_2) - p_1(x_2)\}$$

where $h(t) \leq_x p(x) \triangleq \exists_{v, X} [h(t) = \langle v, X \rangle \wedge v \in p(x) \wedge x \in X]$, provides the element evolutions, dealing with the population p of the application model, which involve object type x at t .

- Changed element evolutions, involving the types from the meta model:

$$\{h[v]_t \mid h \in H - H_{pop} \wedge h(t) \in p_1(x_1) - p_2(x_1) \wedge v \in p_2(x_2) - p_1(x_2)\}$$

Note: $x_1 = x_2$ in this case.

and further, where:

1. $p_2 \triangleq \mathbb{U}_\infty[\text{ADD ONm}(x_1): d_1] (\mathbb{U}_\infty[\text{DELETE ONm}(x_2): d_2] (p_1, e), e)$, and p_1 is the population of the disclosure schema of H at t ,
2. $h[v]_t$ is an abbreviation for: $\lambda s. \text{if } s \leq t \text{ then } h(t) \text{ else } v \text{ fi}$,
3. and Typing is defined as above.

7.4.2 Queries

Some transactions, for instance report generating transactions, may involve queries on the application model. The semantics of Elisa-D information descriptors was defined in the previous chapter, these semantics are connected to the semantics of transactions as queries, in this subsection.

The result of a query must be communicatable, i.e. the (possibly abstract) values in the result of the query must be denoted by label values. This denotation mechanism is based on the identification rules for instances from (abstract) object types provided by the Ident and Order functions, discussed in subsection 6.5.4. This mechanism provides a naming (denotation) convention for the instances of all object types:

$$\text{StdName} : \text{POP} \times \Omega \rightarrow \text{Constant Denotation}$$

where StdName stands for ‘standard name’. See [Hof93] for a more detailed discussion on this naming convention. The StdName function can be extended to the result of a path expression as:

$$\text{StdName} : \text{POP} \times \Omega_{\text{PE}} \rightarrow \mathbb{C}(\text{Constant Denotation})$$

which is identified by:

$$\text{StdName}(\text{Pop}, X) \triangleq \{ \llbracket \text{StdName}(\text{Pop}, v) \uparrow^q \mid \langle v, w \rangle \in^q X \rrbracket \}$$

The semantics of the syntactic category Query is specified by the valuation function:

$$\mathbb{L} : \mathcal{T} \times \text{Query} \times \text{POP} \times \text{ENV} \rightarrow \mathbb{C}(\text{Constant Denotation})$$

$$\mathbb{L}_t[\text{LIST } P] (\text{Pop}, e) \triangleq \text{StdName}(\text{Pop}, \mu_t[\mathbb{D}[P]] (e)) (\text{Pop})$$

The standard names are used as a substitution mechanism to transform (abstract) values into concrete label values in the semantics of the LIST-statement. The effect of the LIST-statement is to properly list these values.

7.4.3 Transaction Semantics

The Elisa-D transactions are properly introduced in terms of assignments, updates and queries in this subsection. The semantics of transactions is provided by the function:

$$\mathbb{T} : \mathcal{T} \times \text{Transaction} \times \text{STATE} \rightarrow \text{STATE}$$

The STATE set is formed by four components:

$$\text{STATE} \triangleq \mathcal{AMH} \times \text{ENV} \times \text{INPUT} \times \text{OUTPUT}$$

where $\text{INPUT} \triangleq \text{Constant Denotation}^*$ provides the (user) input of the information system, and $\text{OUTPUT} \triangleq \mathbb{C}(\text{Constant Denotation})$ holds the output of the information system. The semantics of transactions is expressed in terms of updates and queries as:

$$\begin{aligned} \mathbb{T}_t[a] (\langle H, e, i, o \rangle) &\triangleq \langle H, \mathbb{A}_t[a] (\text{Pop}, e), i, o \rangle \\ \mathbb{T}_t[u] (\langle H, e, i, o \rangle) &\triangleq \langle \mathbb{U}_t[u] (H, e), e, i, o \rangle \\ \mathbb{T}_t[q] (\langle H, e, i, o \rangle) &\triangleq \langle H, e, i, o \uparrow \langle \mathbb{L}_t[q] (H, e) \rangle \rangle \\ \mathbb{T}_t[\text{READ } w] (\langle H, \langle M, V \rangle, i, o \rangle) &\triangleq \langle H, \langle M, V \odot w : \text{Sqr} \{ \llbracket \text{Head}(i) \rrbracket \}, \text{Tail}(i), o \rangle \\ \mathbb{T}_t[t_1; t_2] (\langle H, e, i, o \rangle) &\triangleq \mathbb{T}_t[t_2] (\mathbb{T}_t[t_1] (\langle H, e, i, o \rangle)) \end{aligned}$$

where a is an Assignment, u is an Update, q a Query, and w a Variable. The transaction semantics provided by \mathbb{T} can now be used to define the $\llbracket m \rrbracket_t$ relation for Hydrae; for simple tasks, only consisting of one Elisa-D transaction, this would lead to:

$$H \llbracket m \rrbracket_t H' \triangleq \mathbb{T}_t \llbracket m \rrbracket (\langle H, e, i, \emptyset \rangle) = \langle H', e', i', o \rangle$$

where e denotes the initial environment (containing the default macro declarations), and i the input from the user. For complex task structures, these semantics will have to be constructed from this semantics, in the same way as for Hydra ([Hof93]).

Finally, the depends relation can be refined for transactions. For updates, dependency is derived from the evolution dependency for information descriptors: P , and $\text{ONm}(x) : d$:

$$\begin{aligned} P \text{ Depends } x &\vdash \text{LET } m \text{ BE } P \text{ Depends } x \\ P \text{ Depends } x &\vdash \text{LET } m \text{ ALWAYS BE } P \text{ Depends } x \\ P \text{ Depends } x &\vdash \text{ASSIGN } P \text{ TO } w \text{ Depends } x \\ \text{ONm}(o) : d \text{ Depends } x &\vdash \text{ADD ONm}(o) : d \text{ Depends } x \\ \text{ONm}(o) : d \text{ Depends } x &\vdash \text{DELETE ONm}(o) : d \text{ Depends } x \\ \text{ONm}(o) : d \text{ Depends } x &\vdash \text{CHANGE ONm}(o) : d \text{ TO ONm}(o') : d' \text{ Depends } x \\ \text{ONm}(o) : d \text{ Depends } x &\vdash \text{CHANGE ONm}(o') : d' \text{ TO ONm}(o) : d \text{ Depends } x \\ P \text{ Depends } x &\vdash \text{LIST } P \text{ Depends } x \end{aligned}$$

7.5 Epilogue

This chapter has defined the syntax of Hydrae, and provided an extension of Elisa-D by transactions, relating Hydrae and Elisa-D. The syntax of Hydrae comprised the definition of a proper universe, and a set of well-formedness rules for versions in this universe, resulting in a definition of the IsActMod predicate.

The transactions of Elisa-D, and Hydrae, can be used to update the application model history, so that when an application model evolves, the changes must be communicated to the EIMS by update requests formulated as Hydrae task structures. In the case where the population is changed, this is rather straightforward. For instance, the deletion of all songs recorded on the Record ‘Brothers in Arms’ can be specified as:

```
REPEAT
  LET  $w$  BE A Song recorded on Record: 'Brothers in Arms'
  DELETE Recording:  $w$ , 'Brothers in Arms'
UNTIL NOT SOME Song recorded on Record: 'Brothers in Arms'
```

The REPEAT ... UNTIL clause is presumed to be the textual denotation of a task structure with two subtasks, one decision point, and a loop providing the repetition.

Remark 7.5.1

In a concrete EIMS, one may easily extend the Hydrae with (dangerous!) intentional delete statements. In this case, the above deletion could be abbreviated to:

```
DELETE Song recorded on Record: 'Brothers in Arms'
```

□

The specification of updates for the other components of the application model, however, is more difficult. Consider for example the addition of a series of new object types, or the addition of Hydrae task structures. Such an operation would require a series of ADD statements, adding every single component. When the information structure needs to be modified, or an action specification in the action model must be changed, it is not advisable to do this using a (large number of) traditional ADD or CHANGE updates. Rather, one would like to specify such a modification using a CASE tool interface, and even *test* the modification envisioned in some way before committing it to the evolving information system. This *test* might consist of a prototyping phase, to assess the change to the application model present in the evolving information system actually desired.

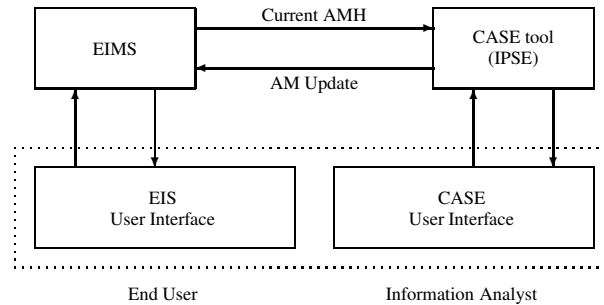


Figure 7.8: CASE and EIMS integration

The integration between a CASE tool (IPSE) and an EIMS is illustrated in figure 7.8. When a change to the application model is needed, the current (or older) version of the application model history (minus the population) is transferred to the repository of the CASE tool. The modifications to the (most recent) application model are made through the CASE tool, and on completion communicated to the EIMS. See [Pro93] for more details on the relation between evolving information systems and CASE tools.

Finally, the four phases of applying the general theory can also be identified for Hydrae, even though they have not played a central role in this chapter.

1. *Identify the information structure and/or application model universe.*

The Hydrae universe is defined in section 7.3, there we restricted ourselves to the syntax of Hydrae.

2. *Prove the correctness of the identified universe with respect to the general theory.*

Since we limited ourselves to the definition of the syntax, it is not possible to provide such a proof, however, the application model universe axioms should be considered as extra constraints on changes to the application model history brought about by the execution of Hydrae tasks.

3. *Are extra, technique dependent, well-formedness rules needed?*

These axioms have been provided by the `IsActMod` predicate, and a refinement of the `PartOf` relation, as defined in subsection 7.3.2.

4. *Interpret and apply the “theorems for free”.*

The theorems for free focus on data modelling aspects, and are therefore not applicable to Hydrae.

Chapter 8

Stratified Hypermedia as a Disclosure Mechanism

It is an important and popular fact that things are not always what they seem. For instance, on the planet Earth, man had always assumed that he was more intelligent than dolphins because he had achieved so much - the wheel, New York, wars and so on - whilst all the dolphins had ever done was muck about in the water having a good time. But conversely, the dolphins had always believed that they were far more intelligent than man - for precisely the same reasons.

From: "The Hitch Hiker's Guide to the Galaxy",
Douglas Adams, Pan Books Ltd.

8.1 Introduction

Users of traditional information systems already find it difficult to maintain an overview of the structure of the stored information, it is even harder to maintain an overview of all data stored in the application model history of an evolving information system. The information analyst has to deal with information structures large enough to be used as wallpaper (even for medium sized modelling problems). When formulating queries, a user may get a feeling of being lost in *conceptual space*, these problems, of being lost in *conceptual space*, or of not being able to get an overview of the stored information, form a direct threat for the first requirement of an evolving information system, see subsection 1.3.1.

Due to the evolution of the underlying information structures, these problems become even more pressing in an evolving information system. A mechanism to tackle these problems is given in this chapter. The basic idea is to build a hypertext browser for (evolving) information systems ([BPW93], [Pro93], [HPW94b]). Such a browser would provide a means of supporting the user and information analyst in the formulation of queries, leading to better support of information disclosure. This browser features a *query by navigation* mechanism, i.e. it allows a query to be built whilst navigating through the information structure. Similar browsers have proved their usefulness in CASE tools ([Big88], [GS90], [Hag92]). The architecture of such systems is sketched in figure 8.1. The arrows in this diagram represent the information flows between the components of the complete system.

Query by navigation is also fruitfully employed in information retrieval systems ([BW92b], [BBB91], [Bru93]). Using such a mechanism in (evolving) information systems, further bridges the gap between information retrieval and (evolving) information systems (see subsection 1.2.2).

Without a query by navigation system, whenever a user wants to know something about objects in the population

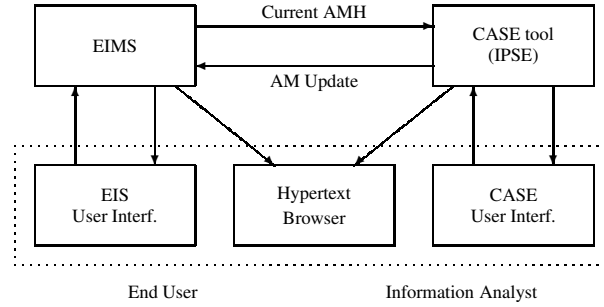


Figure 8.1: Hypertext Browser for an EIMS and CASE tool

of the information system, it is very likely that they will have problems formulating an adequate query. As a consequence, they will retrieve irrelevant (or even wrong) objects and may miss relevant objects. Retrieving irrelevant objects leads to low *precision*, missing relevant objects has a negative impact on *recall* ([SM83]). These terms stem from the information retrieval world, but are equally applicable to information systems.

In this chapter, an EVORM disclosure schema is represented as a *stratified hypermedia architecture*, leading to an implementation of the concept of query by navigation for EVORM. Stratified hypermedia architecture ([BW92b]) features a descriptive layer, the *hyperindex*, comprising a hypertext of characterisations, indexing the lower layer, the *hyperbase*. The hyperbase contains the actual information. Stratified hypermedia architecture offers, by separating description and instantiation, the possibility to formulate information needs by an interactive process of navigation through the descriptive layer.

8.2 Stratified Hypermedia Architecture

Stratified hypermedia architecture, in its simplest form, is a *two level hypermedia architecture* as introduced in [BW92b]. A stratified hypermedia architecture supporting multiple layers of abstraction is discussed in [SDBW91]. The stratified hypermedia architecture features a descriptive layer (hyperindex), indexing the lower layer (the hyperbase). The hyperbase contains the actual information, whereas the hyperindex only provides an outline of the stored information.

An advantage of this architecture is that searchers can navigate within the upper, descriptive, layer to a description of their information need and then transfer to the lower layer via interlayer navigation. This process of *intra* and *inter layer navigation*, we referred to as *query by navigation*. Formally, a layer is a structure $\mathbb{H} \triangleq \langle \mathbb{F}, \mathbb{N}, \mathbb{G}, \mathbb{V} \rangle$ where:

1. \mathbb{F} is a set of information fragments, called the *fragment base*.
2. \mathbb{N} is a set of presentation units (or nodes), referred to as the *node base*.
3. \mathbb{G} is a structure $\langle \mathbb{E}, \mathbb{P} \rangle$, where \mathbb{E} is a set of syntactic categories, and \mathbb{P} is a set of context-free production rules. \mathbb{G} is referred to as the *schema* of the layer.
4. \mathbb{V} is a set of views, called the *mask*.

Fragments are elementary parts of the stored information which can not be decomposed structurally into smaller components. Nodes are units of presentation and are used to present structural elements to the user. The grammar \mathbb{G} is used to structure the information in a layer. Usually, this grammar is provided as a context-free grammar. A view, in its turn, is a structure $\mathbb{V} \triangleq \langle \mathbb{S}, \omega, \mathbb{M}, \pi, \mathbb{L} \rangle$ where:

1. $\mathbb{S} \in \mathbb{E}$ is the *start symbol*

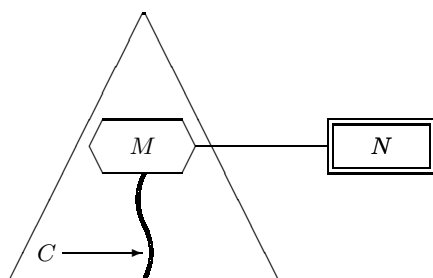


Figure 8.2: The context C of molecule M and presentation by node N

2. \mathbb{M} is the set of subtrees of parse trees generated from \mathbb{S} using \mathbb{G} . Such a subtree, corresponding to a linear path expression, is also called a *molecule*.
3. ω is a relation of $\mathbb{M} \times \mathbb{M}$, such that when $\langle x, y \rangle \in \omega$ parse tree x is a subtree of parse tree y . ω is referred to as the actual structure.
4. $\pi : \mathbb{M} \rightarrow \mathbb{N}$ maps each molecule from \mathbb{M} to a presentation unit, a *node*.
5. \mathbb{L} is a set of *associative link*.

The basic concept in the interaction with the searcher is a *context*. A context in the hyperindex represents (part of) the information need of the searcher. Figure 8.2 shows the outline of a parse tree. A path from some terminal in a parse tree to another vertex in the parse tree (terminal or non-terminal) is called the context of that molecule. Note that every vertex in a parse tree is the root of one subtree of this parse tree, and thus every vertex in the parse tree has associated a molecule. By operating on the context (enlarge/refine), the searcher will reach a point in her search where no improvement is possible. The searcher will then store this context for the time being, and start a new search for the missing part of the information need.

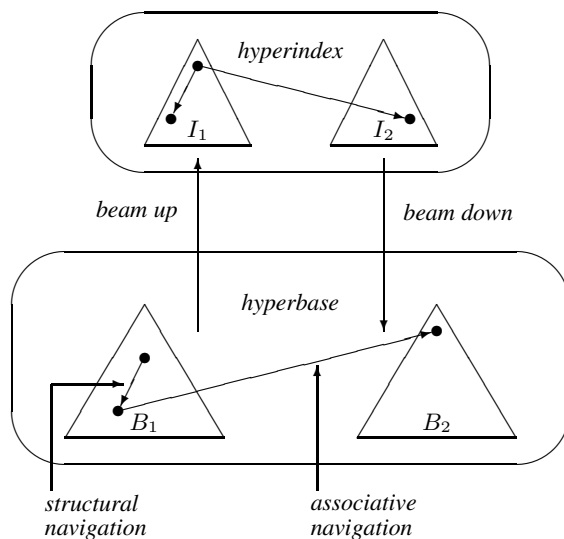


Figure 8.3: Operations of the hypermedia

Two kinds of navigations between molecules are presented in figure 8.3, two parse trees of the hyperbase layer are presented (B_1, B_2), and two parse trees of the hyperindex layer (I_1, I_2). The movement from one molecule to another, using the underlying structure of a parse tree, is called *structural navigation*. Selecting an associative link

initiates traversal of such a link, called *associative navigation*, leads to a change in context (parse tree). Associative links are used to feature cross-references between a fragment in one molecule, and a fragment in another molecule. The beam up and beam down operations are used to facilitate inter layer navigation.

During the quest for the fulfillment of their information need, searchers will usually have activated a number of contexts, one of which is selected as the *focus* for further processing. This set of activated contexts is called the *guide* ([SDBW91]), leading to the eventual query of the user.

8.3 Exploring the Disclosure Schema

Before formally describing the data modelling technique EVORM as a stratified hypermedia architecture, we present a demonstration of the benefits of the resulting query by navigation system. The examples show how the system supports the formulation of queries.

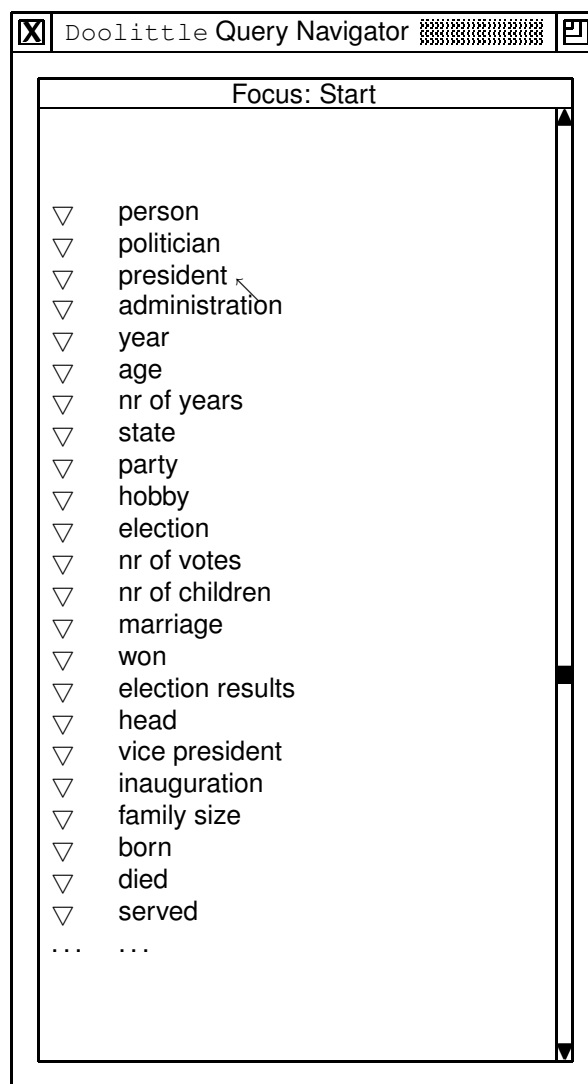


Figure 8.4: The starting node

The process of query formulation corresponds to a search through the information system to fulfill some information need. The request of the searcher is formulated by stepwise refining or enlarging the current description (the *focus*)

of this need, until the searcher recognises the current description as the best possible description of (a part of) the information need. Usually, the final description will result from combining a set of such descriptions (the guide) using more complex Elisa-D operations.

The examples in this section, provide a general idea of what searchers have to do subsequently, and what screens they will encounter, when formulating their information need. Each screen contains, in its header, the current focus, and in its body the direct environment of the current focus. In the examples, we limit ourselves to the presidential database depicted in figure 6.8. Normally, the entire disclosure schema will be represented in the hyperindex.

The first example, figure 8.4, shows the screen which corresponds to the starting point of a search in which the searcher has not yet revealed anything. This screen is referred to as the *starting node* of the system. The starting node contains all object types of the disclosure schema under consideration. The searcher can now choose one of the objects as the focus for further processing. This selection is the first refinement of the searcher's information need. A selection of an item in a node is denoted by the symbol \nwarrow in the figures. The symbol ∇ is a button for a refinement step, while \triangle is used for an enlargement step. Finally, \triangleright is the button used for an associative link.

In the second example, figure 8.5, the searcher wants to find those presidents married to someone involved in politics. The search begins with the starting node (see figure 8.4). We describe only one path leading to a descriptor of her information need, in which the searcher is directly heading for the goal, without any backtracking. The searcher starts with selecting president as the focus. The associated screen shows the direct environment of president. Then the searcher selects president involved in marriage as the next focus. In the resulting screen the searcher continues with president involved in marriage with person. The screen associated with this focus shows the sentence president involved in marriage with politician, which is a proper description of the original information need. Now the searcher can satisfy her information need, by demanding the result of the request via a beam down. The system will present the answer in a standard tabular format, or in the following more verbose form:

president x involved in marriage with politician y

where x denotes a president and y a politician.

In the third example, figure 8.6, the searcher is interested in the birth year and age of death of some person. So the user will select Person from figure 8.4, resulting in the first screen of figure 8.6. Since the birth year and age of death are only recorded for presidents, the searches will not find any continuation of the current focus concerned with birth years. At this point, the searcher decides to limit her information need to presidents, and selects (the subtype) President. Now the searcher can refine her focus by choosing either president having been born or president having died as the focus for further processing. When selecting the former sentence as focus, using A , the sentence president having been born in year appears. This satisfies the first part of the information need. To satisfy the second part, the searcher must create another context. She now focuses, using B , on the age at death of persons. After choosing president having died at age, she has finally fulfilled her information need, and can request the result of this guide. The system will present all answers in the following verbose way:

president x having been born in year y , and having died at age z

where x denotes a president, y a birth year and z an age of death.

The queries resulting from the query by navigation process described above are rather simple, and correspond to linear information descriptors (with an underlying linear path expression) through the disclosure schema. When a structure editor is added to Elisa-D queries, depicted in figure 8.7, these simple queries can be combined into more complex ones, utilising the expressive power of Elisa-D to its fullest. The process of building a query in such a structure editor, is referred to as *query by construction*.

8.4 The EVORM Hyperindex Layer

To represent EVORM disclosure schemas (spanned by \mathcal{O}_{ds}) as a stratified hypermedia architecture, a disclosure schema is divided into two levels of abstraction first. The concrete level consists of all concrete object types, i.e. label types. The non-concrete object types (the *defoliated information structure*, see [BW92a]) form the abstract level. This level thus consists of the following components:

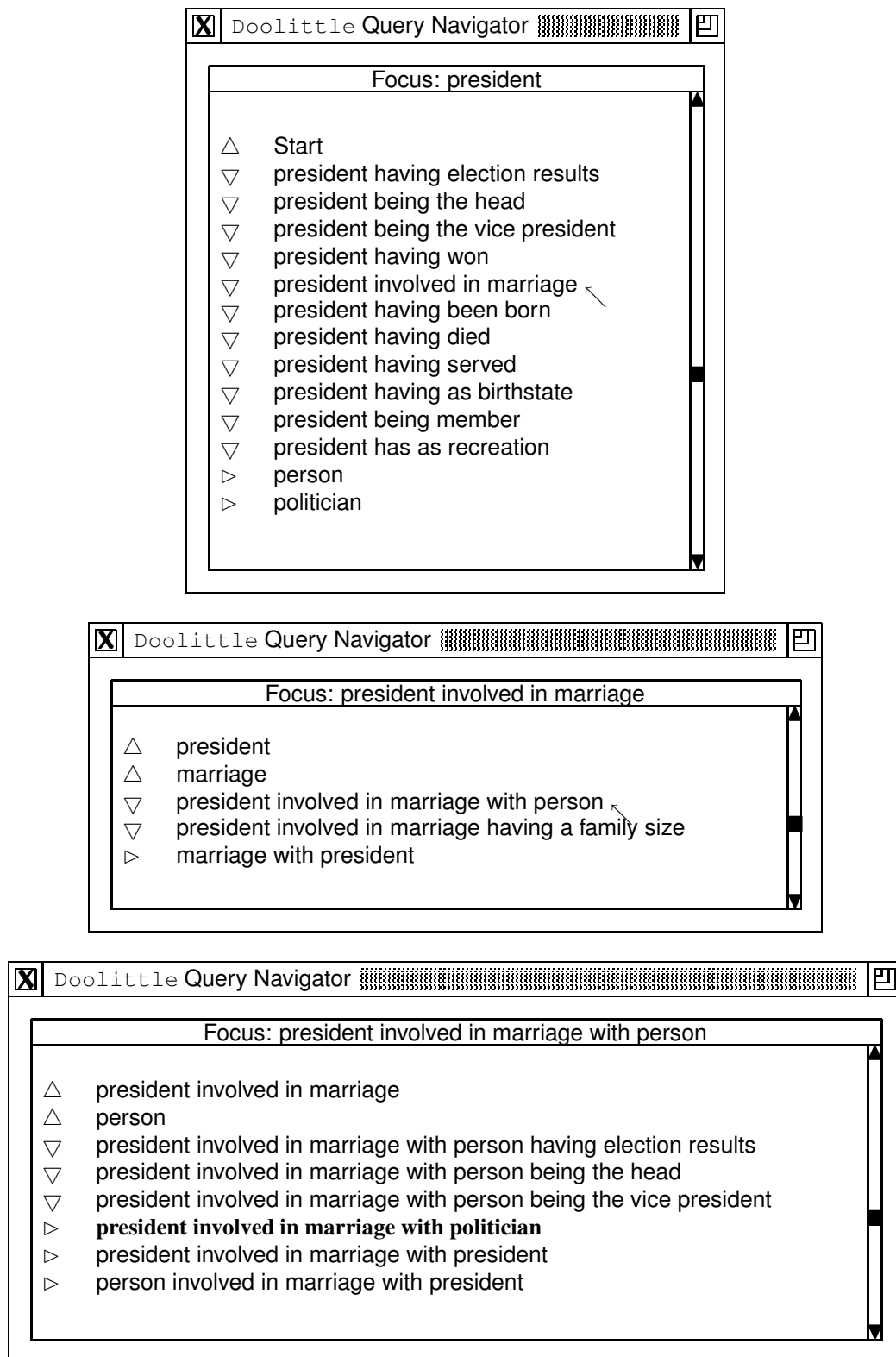
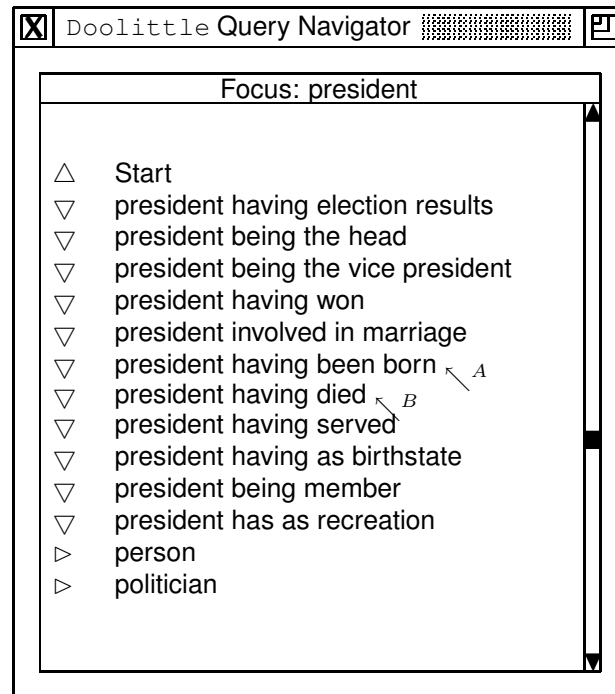
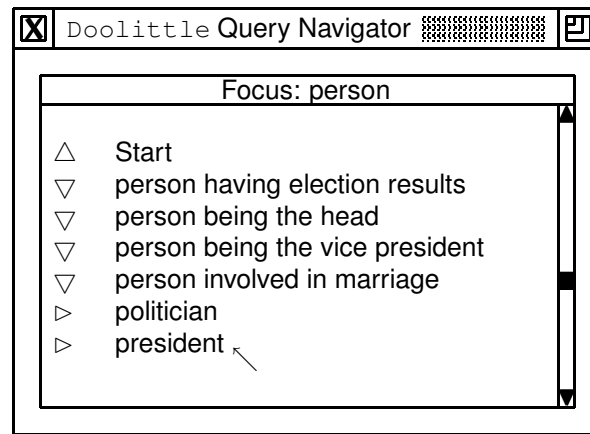


Figure 8.5: The quest for a president who is married with a politician



A:

B:

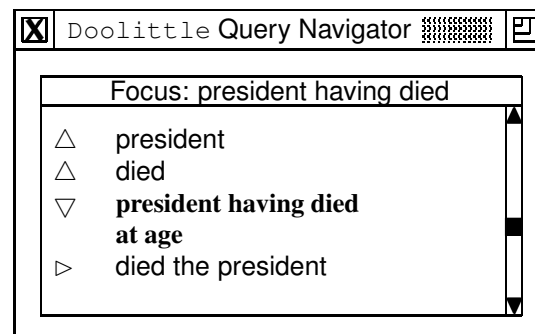
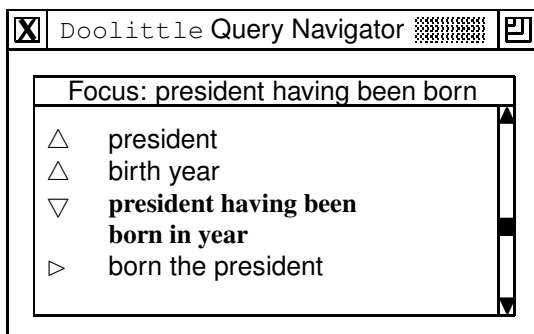


Figure 8.6: The quest for a persons birth year and age of death

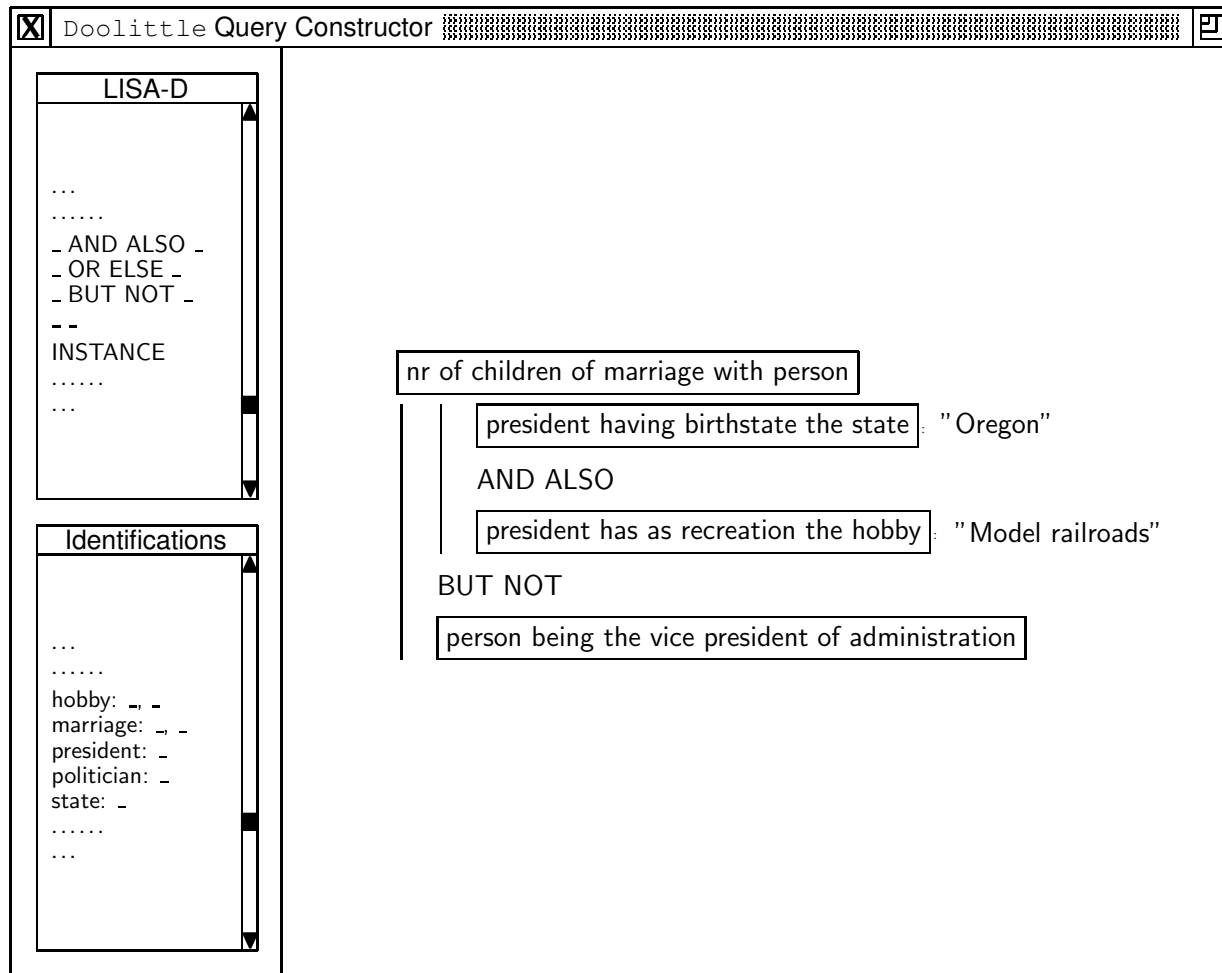


Figure 8.7: Query by construction

- entity types (\mathcal{E}_{ds}), fact types (\mathcal{F}_{ds}), power types (\mathcal{G}_{ds}), sequence types (\mathcal{S}_{ds}), schema types (\mathcal{C}_{ds})
- generalisation (Gen), specialisation (Spec)

These two levels of abstraction can be transformed into the layers constituting a stratified hypermedia architecture (figure 8.8). The hyperbase is constructed from the instantiation of an information structure (its population) in the next section, in this section the focus is on the hyperindex.

Object role models allow for the verbalisation of queries by path expressions, built from names of object types and role names (see chapter 6). These path expressions are used to construct the hyperindex. In this chapter, we restrict ourselves to linear paths expressions. Linear path expressions are constructed by concatenating object types $O \in \mathcal{O}$, predicates $p \in \mathcal{P}$, and reverse predicates p^- . Predicates and object types are the elementary parts from which the set of *linear path expressions* (\mathcal{PE}_{lin}) is constructed. These linear path expressions will play a central role in the introduction of the hyperindex for EVORM disclosure schemas.

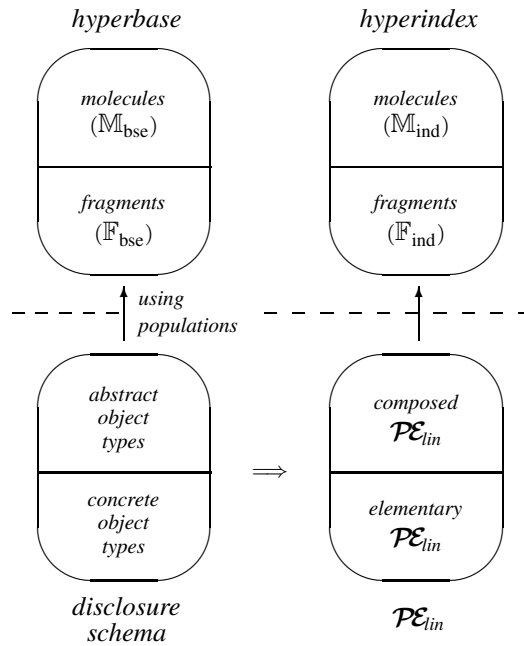


Figure 8.8: Transforming a PSM structure into a stratified hypermedia structure

Formally, a hyperindex is introduced as a structure $\mathbb{H}_{ind} \triangleq \langle \mathbb{F}_{ind}, \mathbb{N}_{ind}, \mathbb{G}_{ind}, \mathbb{V}_{ind} \rangle$. Such a hyperindex can be associated to each disclosure schema spanned by the set of object types \mathcal{O}_{ds} . The components of the hyperindex are now introduced successively.

8.4.1 Fragment base

The fragment base (\mathbb{F}_{ind}) of the hyperindex, simply contains the elements for verbalisations of linear path expressions. So we have:

$$\mathbb{F}_{ind} \subseteq \text{Names}^+$$

Some examples from the fragment base of the presidential schema example are:

President, Marriage, involved in, Election results

8.4.2 Node base

Next, we focus on the construction of the hyperindex node base (\mathbb{N}_{ind}). Navigation through the defoliated information structure corresponds to the construction of a linear path expression. This path expression represents this search process. The nodes correspond to the intermediate results in the search process.

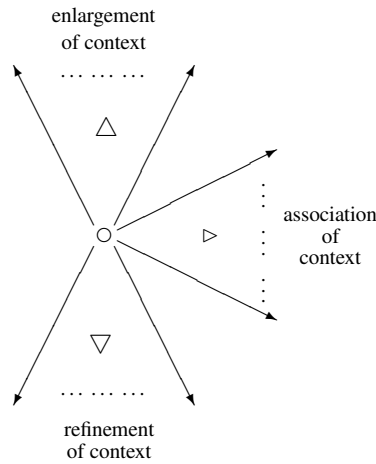


Figure 8.9: The environment of a node

A node itself consists of three components, the currently selected linear path expression, a set of less specific path expressions, a set of more specific path expressions, and a set of path expressions reachable by an associative link. The latter three components describe the environment of the current focus in the disclosure schema, depicted in figure 8.9. This leads to the following demarcation for the nodes in the hyperindex:

$$\mathbb{N}_{\text{ind}} \subseteq \mathcal{PE}_{\text{lin}} \times \wp(\mathcal{PE}_{\text{lin}}) \times \wp(\mathcal{PE}_{\text{lin}}) \times \wp(\mathcal{PE}_{\text{lin}})$$

In general, a node $\langle c, \{u_1, \dots, u_l\}, \{d_1, \dots, d_m\}, \{a_1, \dots, a_n\} \rangle$ leads to a presentation depicted in figure 8.10. The function ρ is used to verbalise linear path expressions, and will be introduced at the end of this section.

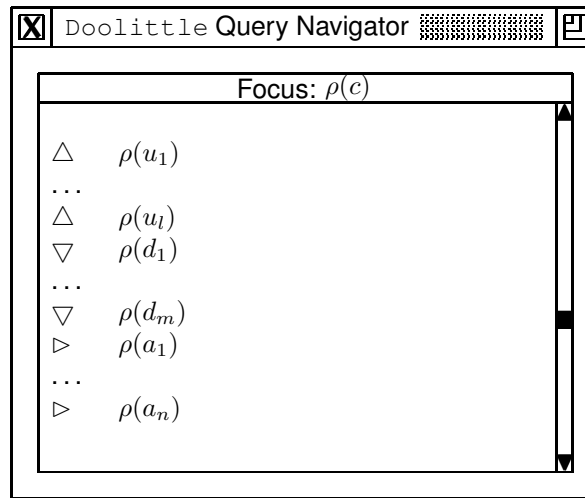


Figure 8.10: Presentation of focus C and its direct environment

8.4.3 Schema

The following step in defining the hyperindex is the introduction of its schema:

$$\mathbb{G}_{\text{ind}} \triangleq \langle \mathbb{E}_{\text{ind}}, \mathbb{P}_{\text{ind}} \rangle$$

In our approach, we navigate through the hyperindex by refinements and enlargements of a linear path expression, corresponding to a molecule in the hyperindex. An alternative approach would be to define a grammar for the verbalisation of these path expressions, and navigate through the verbalisations rather than the underlying (semantics) linear path expression.

The context-free production rules for the hyperindex (\mathbb{P}_{ind}), define the way in which linear path expression can be extended. For any $p \in \mathcal{P}_{ds}$ and $x \in \mathcal{N}_{ds}$, the following rules belong to \mathbb{P}_{ind} :

$$\begin{aligned} \langle P \rangle &\rightarrow x \\ \langle P \rangle &\rightarrow \langle P \rangle \circ p \circ \text{Fact}(p) \\ \langle P \rangle &\rightarrow \langle P \rangle \circ p^{\leftarrow} \circ \text{Base}(p) \end{aligned}$$

The linear paths in the above format, also accommodate objectification. In the molecules, the objectification of fact types leads to different ways for refinement of the current focus:

- the objectified fact type can be used as an entity type, abstracting from the composition into predicates,
- the decomposition of the objectified fact type into predicates can be taken into account.

In figure 8.6 this resulted in `president involved in marriage having family size`, and `president involved in marriage with person`, respectively. For the hyperindex, we consider $\langle P \rangle$ as the only syntactic category, so: $\mathbb{E}_{\text{ind}} \triangleq \{\langle P \rangle\}$.

8.4.4 Views

The hyperindex for an EVORM disclosure schema, contains only one single view. A view is formally introduced as a structure:

$$\mathbb{V}_{\text{ind}} \triangleq \langle \mathbb{S}_{\text{ind}}, \omega_{\text{ind}}, \mathbb{M}_{\text{ind}}, \pi_{\text{ind}}, \mathbb{L}_{\text{ind}} \rangle$$

The starting point of this view is $\mathbb{S}_{\text{ind}} \in \mathbb{E}_{\text{ind}}$, necessarily being $\langle P \rangle$. The molecules \mathbb{M}_{ind} are formed by the set of linear path expressions. The actual structure ω_{ind} is a subset of $\mathbb{M}_{\text{ind}} \times \mathbb{M}_{\text{ind}}$ that describes the set of linear path expressions that are possible in the disclosure schema under consideration. This set is identified by:

$$\begin{aligned} \omega_{\text{ind}} &\triangleq \{ \langle \epsilon, x \rangle \mid x \in \mathcal{N}_{ds} \} \\ &\cup \{ \langle P x, P x \circ r \circ \text{Fact}(r) \rangle \mid P x \in \mathbb{M}_{\text{ind}} \wedge x \in \mathcal{N}_{ds} \wedge r \in \mathcal{P}_{ds} \wedge \text{Base}(r) \xrightarrow{\sim} x \} \\ &\cup \{ \langle P x, P x \circ r^{\leftarrow} \circ \text{Base}(r) \rangle \mid P x \in \mathbb{M}_{\text{ind}} \wedge x \in \mathcal{N}_{ds} \wedge r \in \mathcal{P}_{ds} \wedge \text{Fact}(r) \xrightarrow{\sim} x \} \end{aligned}$$

where ϵ denotes the (textually) empty path expression. The presentation of molecules by π_{ind} is covered in the next subsection. The associative links (\mathbb{L}_{ind}) of the hyperindex layer are used to handle specialisation and generalisation occurring in the disclosure schema, and the reversal of the current focus. They are defined in the following way:

- If $x \text{ Spec } y$, then we provide an associative link from each molecule that contains x (as last occurrence) in its header, to the molecule in which this x is replaced by y .
This replacement is executed by the \triangleright entries in the node.
- If $x \text{ Gen } y$ and the header contains x as last occurrence, then a special entry is included where x is replaced by y .

- To any path expression, the reverse of the current path expression can be associated.

Normally, a focus is enlarged or refined by operating on the tail of the current focus. Sometimes, however, one may want to operate on the head of focus, and for this purpose the reversal of the currently focussed path expression can be employed.

Let x, y be object types, and let P be a linear path expression, then the set of associative links is formally identified by:

$$\mathbb{L}_{\text{ind}} \triangleq \{ \langle P x, P y \rangle \mid P x, P y \in \mathbb{M}_{\text{ind}} \wedge y \rightsquigarrow x \} \cup \{ \langle P, \text{Rev}(P) \rangle \mid P \in \mathbb{M}_{\text{ind}} \wedge P \neq \text{Rev}(P) \}$$

where Rev is recursively defined as:

$$\begin{aligned} \text{Rev}(P \circ p \circ x) &\triangleq x \circ p^{\leftarrow} \circ \text{Rev}(P) \\ \text{Rev}(P \circ p^{\leftarrow} \circ x) &\triangleq x \circ p \circ \text{Rev}(P) \\ \text{Rev}(x) &\triangleq x \end{aligned}$$

An example of such a reversal is:

$$\text{Rev}(x \circ p \circ f \circ q^{\leftarrow} \circ y) = y \circ q \circ f \circ p^{\leftarrow} \circ x$$

To demonstrate the differences between specialisation and generalisation, figure 8.11 shows a part of the hyperindex located around a starting molecule.

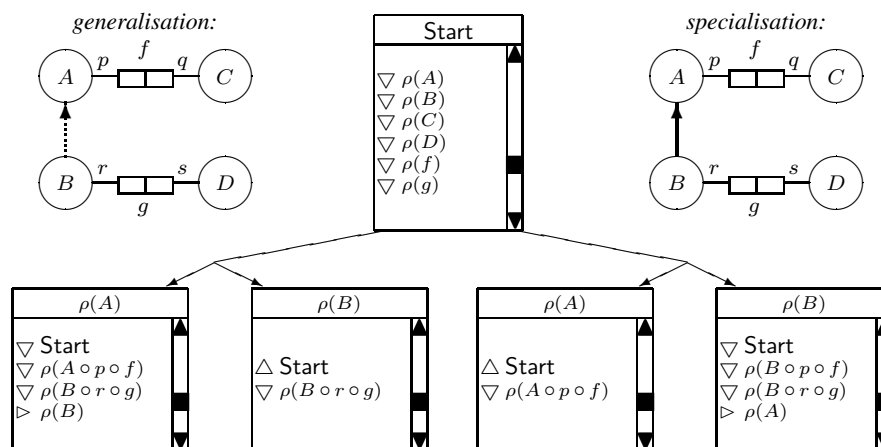


Figure 8.11: Hyperindex dealing with generalisation and specialisation

8.4.5 Presentation of molecules

Molecules are presented by nodes specified by π_{ind} . As stated before, a molecule will be presented by a molecule containing the direct environment of the molecule. Therefore, the presentation of a molecule M is identified as:

$$\pi_{\text{ind}}(M) \triangleq \langle M, M_{\Delta}, M_{\nabla}, M_{\triangleright} \rangle$$

where the direct environment of M is captured by:

$$\begin{aligned} M_{\Delta} &\triangleq \{ N \mid \langle N, M \rangle \in \omega_{\text{ind}} \} \\ M_{\nabla} &\triangleq \{ N \mid \langle M, N \rangle \in \omega_{\text{ind}} \} \\ M_{\triangleright} &\triangleq \{ A \mid \langle M, A \rangle \in \mathbb{L}_{\text{ind}} \} \end{aligned}$$

All that remains to be done with respect to the presentation of the molecules, is a proper definition of ρ . This can be done by a set of derivation rules, with an associated preference (using penalty points). The predicate $\rho(P, n, x)$ is employed to denote that path expression P is verbalised as n , with penalty x . In the derivation of a verbalisation of a linear path expression, we employ the names provided by the user, using the naming functions, as much as possible. The name of the empty path expression is defined by:

[Verb1] (*empty path expression*) $\vdash \rho(\epsilon, \text{Start}, 0)$

The names provided by the information analyst have the highest preference:

[Verb2] (*connector naming*) $\text{Connector}(p, q) = n \vdash \rho(p \circ \text{Fact}(p) \circ q^{\leftarrow}, n, 0)$

[Verb3] (*predicator naming*) $\text{PNm}(p) = n \vdash \rho(p, n, 0)$

[Verb4] (*reverse predicator naming*) $\text{RNm}(p) = n \vdash \rho(p^{\leftarrow}, n, 0)$

[Verb5] (*object type naming*) $\text{ONm}(x) = n \vdash \rho(x, n, 0)$

The lexicon is used to derive default verbalisations for implicit, or other unnamed, object types. The predefined names in the lexicon have a lower preference in the verbalisation than ‘user’ defined names. Let $\text{UserDef}(n, P)$ be a predicate stating that name n is a user defined name with definition P , so in particular $\text{Lexicon}(n, P)$. Then the defined names of the lexicon are used for verbalisations as follows:

[Verb6] (*lexicographic verbalisation 1*) $\text{UserDef}(n, P) \vdash \rho(P, n, 0)$

[Verb7] (*lexicographic verbalisation 2*) $\text{Lexicon}(n, P) \vdash \rho(P, n, 1)$

Composed linear path expressions are verbalised by the following rule:

[Verb8] (*concatenation of verbalisations*)

$$\rho(P_1, n_1, \alpha_1) \wedge \rho(P_2, n_2, \alpha_2) \vdash \rho(P_1 \circ P_2, n_1 \ n_2, \alpha_1 + \alpha_2 + 1)$$

As an example, consider the following names for figure 8.11.

$\text{ONm}(A) = \text{Person}$	$\text{PNm}(p) = \text{having as}$
$\text{ONm}(B) = \text{President}$	$\text{PNm}(q) = \text{the}$
$\text{ONm}(C) = \text{Hobby}$	$\text{PNm}(r) = \text{being}$
$\text{ONm}(D) = \text{Party}$	$\text{PNm}(s) = \text{of}$
$\text{ONm}(f) = \text{Recreation}$	$\text{Connector}(p, q) = \text{having as his}$
$\text{ONm}(g) = \text{Member}$	$\text{Connector}(r, s) = \text{being a member of the}$

This would lead, for instances, to the following verbalisations:

$$\begin{aligned} &\rho(A \circ p \circ f \circ q^{\leftarrow} \circ C, \text{Person having as Recreation the Hobby}, 4) \\ &\rho(A \circ p \circ f \circ q^{\leftarrow} \circ C, \text{Person having as his Hobby}, 2) \\ &\rho(A \circ p \circ f \circ q^{\leftarrow} \circ C, \text{Person INVOLVED IN Recreation OF Hobby}, 6) \end{aligned}$$

The chosen penalty system in the above derivation rules should be further tuned, by using it in a real implementation. These derivation rules are used to derive a verbalisation with the highest preference (lowest penalty), as follows:

$$\begin{aligned} \text{Cost}(P) &\triangleq \min \{ \alpha \mid \exists n [\rho(P, n, \alpha)] \} \\ \rho(P) &\triangleq n \text{ such that } \rho(P, n, \text{Cost}(P)) \end{aligned}$$

In our example, we would have: $\text{Cost}(A \circ p \circ f \circ q^{\leftarrow} \circ C) = 2$, so the resulting verbalisation would be: Person having as his Hobby. Note that we have omitted the capital letters in the example nodes.

8.5 The EVORM Hyperbase Layer

The extra-temporal population of the disclosure schema is represented in the hyperbase in a verbose way. Linear path expressions were employed as the molecules for the hyperindex. Similarly, for the hyperbase, a richer subset of the path expressions will be used. This subset, the *conjoined path expressions* (\mathcal{PE}_{con}), consists of linear path expressions extended with constants, and conjunctions of such linear path expressions. The constants in the linear path expressions, are used to get a grip on the values in the extra-temporal population. Conjunctions of linear path expressions (P, Q) are formed by the \wedge operator, which is defined as: $P \wedge Q \triangleq (\mathcal{f} P) \cap (\mathcal{f} Q)$.

The translation of an instantiation of an EVORM disclosure schema into a hyperbase (\mathbb{H}_{bse}) is also carried out bottom-up, the fragment base (\mathbb{F}_{bse}) is defined first, followed respectively, by the node base (\mathbb{N}_{bse}), the schema (\mathbb{G}_{bse}) and the views (\mathbb{V}_{bse}).

8.5.1 Fragment base

In chapter 6, the syntactic category `Constant Denotation` was employed to denote both abstract and concrete instances. In the verbalisation of the conjoined path expressions names from `Names`, and constant denotations will be applied for the denotation of instances. This leads to the following restriction of \mathbb{F}_{bse} :

$$\mathbb{F}_{bse} \subseteq (\text{Names} \cup \text{Constant Denotation})^+$$

8.5.2 Node base

A node in the node base simply corresponds to a sentence formulated in terms of the elements from the fragment base. The nodes in the hyperbase, also contain information for the structural and associative navigation, analogous to nodes in the hyperindex (see figure 8.11). Therefore, the node base for the hyperbase is demarcated by:

$$\pi_{bse} \subseteq \mathcal{PE}_{con} \times \mathcal{O}(\mathcal{PE}_{con}) \times \mathcal{O}(\mathcal{PE}_{con}) \times \mathcal{O}(\mathcal{PE}_{con})$$

Note that not all nodes will be used for the representation of the molecules.

8.5.3 Schema

The structure of the hyperbase is described by the grammar $\mathbb{G}_{bse} \triangleq \langle \mathbb{F}_{bse}, \mathbb{P}_{bse} \rangle$. Let Pop_∞ be the extra-temporal population of the disclosure schema, then for any $x, y \in \mathcal{N}_{ds}$ such that $y \rightsquigarrow x$, $i \in \text{Pop}_\infty(x)$, and $p \in \mathcal{P}_{ds}$, the following rules are present in \mathbb{P}_{bse} :

$$\begin{aligned} \langle P_x \rangle &\rightarrow x \circ i \\ \langle P_x \rangle &\rightarrow \langle P_y \rangle \\ \langle P_x \rangle &\rightarrow \langle P_x \rangle \circ p \circ \langle P_{\text{Fact}(p)} \rangle \\ \langle P_x \rangle &\rightarrow \langle P_x \rangle \circ p^{\leftarrow} \circ \langle P_{\text{Base}(p)} \rangle \\ \langle G_x \rangle &\rightarrow \langle P_x \rangle \\ \langle G_x \rangle &\rightarrow \langle G_x \rangle \wedge \langle G_x \rangle \\ \langle G \rangle &\rightarrow \langle G_x \rangle \end{aligned}$$

Note that the above syntax is actually a two level grammar ([WMP⁺76]). The context-free rules (\mathbb{P}_{bse}) specifying the structure of these object classes will form, together with the rules specifying projections and joins, the actual structure ω_{bse} of the hyperbase.

8.5.4 Views

The hyperbase also contains only one single view, describing completely the structure of the hyperbase. The view on the hyperbase is formally introduced as a structure:

$$\mathbb{V}_{\text{bse}} \triangleq \langle \mathbb{S}_{\text{bse}}, \omega_{\text{bse}}, \mathbb{M}_{\text{bse}}, \pi_{\text{bse}}, \mathbb{L}_{\text{bse}} \rangle$$

The starting point of this view is $\mathbb{S}_{\text{bse}} \triangleq \langle G \rangle$. The molecules \mathbb{M}_{bse} are formed by $\mathcal{PE}_{\text{con}}$. The actual structure ω_{bse} is a subset of $\mathbb{M}_{\text{bse}} \times \mathbb{M}_{\text{bse}}$, identified by:

$$\begin{aligned} \omega_{\text{bse}} \triangleq & \{ \langle \epsilon, x \rangle \mid x \in \mathcal{N}_{ds} \} \\ \cup & \left\{ \langle P \ x \ i, P \ x \ i \circ r \circ \text{Fact}(r) \ j \rangle \mid \begin{array}{l} P \ x \in \mathbb{M}_{\text{bse}} \wedge x \in \mathcal{N}_{ds} \wedge r \in \mathcal{P}_{ds} \\ \wedge \text{Base}(r) \rightsquigarrow x \wedge j \in \text{Pop} \cdot \text{Base}(r) \end{array} \right\} \\ \cup & \left\{ \langle P \ x \ i, P \ x \ i \circ r^{\leftarrow} \circ \text{Base}(r) \ j \rangle \mid \begin{array}{l} P \ x \in \mathbb{M}_{\text{bse}} \wedge x \in \mathcal{N}_{ds} \wedge r \in \mathcal{P}_{ds} \\ \wedge \text{Fact}(r) \rightsquigarrow x \wedge j \in \text{Pop} \cdot \text{Fact}(r) \end{array} \right\} \\ \cup & \{ \langle x \ G, x \ G \wedge x \ H \rangle \mid x \ G, x \ H \in \mathbb{M}_{\text{bse}} \} \end{aligned}$$

The presentation of hyperbase molecules by π_{bse} is covered in the next subsection. The associative links (\mathbb{L}_{bse}) of the hyperbase layer are introduced next. The links from \mathbb{L}_{bse} are used to handle specialisation and generalisation occurring in the actual disclosure schema, and are defined as:

$$\begin{aligned} \mathbb{L}_{\text{bse}} \triangleq & \{ \langle G \ x \circ i, G \ y \circ i \rangle \mid G \ x \circ i, G \ y \circ i \in \mathbb{M}_{\text{bse}} \wedge y \rightsquigarrow x \} \\ \cup & \{ \langle G, \text{Rev}(G) \rangle \mid G \in \mathbb{M}_{\text{bse}} \wedge \text{Rev} \downarrow G \} \end{aligned}$$

where Rev is recursively defined as:

$$\begin{aligned} \text{Rev}(P \circ p \circ x \circ i) & \triangleq x \circ i \circ p^{\leftarrow} \circ \text{Rev}(P) \\ \text{Rev}(P \circ p^{\leftarrow} \circ x \circ i) & \triangleq x \circ i \circ p \circ \text{Rev}(P) \\ \text{Rev}(x \circ i) & \triangleq x \circ i \\ \text{Rev}(G_1 \wedge G_2) & \triangleq \perp \end{aligned}$$

Note that the reverse of a guide is not defined. The path expressions in a guide must start out from the same object type (or a type related one). When reversing a path expression, this rule would in general be violated.

8.5.5 Presentation of molecules

The presentation of molecules in the hyperbase is done in an analogous manner to that of the molecules of the hyperindex. The presentation of a molecule M is determined by:

$$\pi_{\text{bse}}(M) \triangleq \langle M, M_{\triangle}, M_{\nabla}, M_{\triangleright} \rangle$$

where the components of the node, capturing the environment of, are defined as:

$$\begin{aligned} M_{\triangle} & \triangleq \{ N \mid \langle N, M \rangle \in \omega_{\text{bse}} \} \\ M_{\nabla} & \triangleq \{ N \mid \langle M, N \rangle \in \omega_{\text{bse}} \} \\ M_{\triangleright} & \triangleq \{ A \mid \langle M, A \rangle \in \mathbb{L}_{\text{bse}} \} \end{aligned}$$

The verbalisation function ρ must be extended with three extra derivation rules for the hyperbase. Using these three extra rules, the nodes of the hyperbase can also be verbalised properly.

[Verb9] (*constant denotations*) $c \in \Omega \vdash \rho(c, \text{StdName}(\text{Pop}, c), 0)$

In the current setup, fact type instances occurring in conjuncted path expressions, will be verbalised twice. Consider the path expression:

$$x \circ i \circ p \circ f \circ j \circ q^{\leftarrow} \circ y \circ k$$

where $f \in \mathcal{F}_{ds}$ and $p, q \in f$. The fact type instance j will, necessarily, be the one for which $j(p) = i$ and $j(q) = k$, so the j does not have to be verbalised separately. This leads to the following rule for verbalisation:

[Verb10] (*fact type instances*) If $f \in \mathcal{F}_{ds}$ and $p, q \in f$, then:

$$\rho(p \circ f \circ q^{\leftarrow}, n, \alpha) \vdash \rho(p \circ f \circ i \circ q^{\leftarrow}, n, \alpha)$$

Note that any alternative verbalisation of $p \circ f \circ i \circ q^{\leftarrow}$ verbalising fact instance i , has always at least one higher penalty due to the extra concatenation ($\circ i$) needed. Finally, conjunctions are verbalised by:

[Verb11] (*conjunction of sentences*) If $x, y \in \mathcal{N}_{ds}$ such that $x \sim y$, and $i \in \text{Pop}(x)$, then:

$$\rho(x \circ i \circ G_1, n_1, \alpha_1) \wedge \rho(G_2, n_2, \alpha_2) \vdash \rho(x \circ i \circ G_1 \wedge y \circ i \circ G_2, n_1, \text{and } n_2, \alpha_1 + \alpha_2)$$

8.6 Inter Layer Navigation

In this section, we relate hyperbase and hyperindex using the beam up and beam down operations. We distinguish two kinds of beam up and down operations. The first kind uses the molecules themselves, being path expressions, as the characterisation of the molecules. This kind leads to *strong beam up* and *strong beam down* functions. The second kind uses the representation, the nodes, as a basis for the characterisation of molecules, leading to *weak beam up* and *weak beam down* functions.

8.6.1 Strong beaming

In the strong beam down operation, the linear path expressions in the guide can be combined into one single path expression by f , $-$, \cup and \cap . For instance, linear path expressions G_1, G_2 and G_3 can be combined into: $f(G_1) \cup (f(G_2) - f(G_3))$. When using a strong beam down from a guide in the hyperindex to molecules in the hyperbase, all molecules in the hyperbase corresponding to instances in the result of the path expression associated to the guide, will be offered to the user. Searchers may then select the final beam down destination from this list.

Formally, the strong beam down operation (BD_s) results, at any point in time, in a set of conjuncted path expressions. These resulting path expressions, must describe the same set of instances as the original path expression resulting from the guide. To this end, we first introduce for any population Pop of the disclosure schema, the following predicate over path expressions:

$$\text{InPop}_t(A, B) \triangleq \mu_t[A](\lambda_{t \in \mathcal{T}}. \text{Pop}) \subseteq \mu_t[B](\lambda_{t \in \mathcal{T}}. \text{Pop})$$

The strong beam down operation can be introduced using this predicate. Let x be an object type, p a predicate, and P, Q linear path expressions, then the strong beam down operation is recursively defined as:

$$\begin{aligned} \text{BD}_s(t, x) &\triangleq \{x \circ i \mid \text{InPop}_t(x \circ i, x)\} \\ \text{BD}_s(t, p) &\triangleq \{p\} \\ \text{BD}_s(t, p^{\leftarrow}) &\triangleq \{p^{\leftarrow}\} \\ \text{BD}_s(t, f(P)) &\triangleq \text{BD}_s(t, P) \\ \text{BD}_s(t, P \circ Q) &\triangleq \{A \circ B \mid A \in \text{BD}_s(t, P) \wedge B \in \text{BD}_s(t, Q) \wedge \text{InPop}_t(A \circ B, P \circ Q)\} \\ \text{BD}_s(t, P \cap Q) &\triangleq \{A \wedge B \mid A \in \text{BD}_s(t, P) \wedge B \in \text{BD}_s(t, Q) \wedge \text{InPop}_t(A \wedge B, P \cap Q)\} \\ \text{BD}_s(t, P - Q) &\triangleq \{A \mid A \in \text{BD}_s(t, P) \wedge \text{InPop}_t(A, P - Q)\} \end{aligned}$$

A direct result from this definition is the following corollary:

Corollary 8.6.1 $P \in \text{BD}_s(t, Q) \Rightarrow \text{InPop}_t(P, Q)$

When beaming up from a molecule in the hyperbase, a proper guide must be reconstructed. To do so, the current focus in the hyperbase is dissected into a set of linear path expressions, corresponding to the new guide. This effectively means that the conjunctions (\wedge) and instances (i) must be filtered from the conjuncted path expression corresponding to the current focus in the hyperbase. Let x be an object type, i be a value from Ω , p a predicate, and P, Q be conjuncted path expressions, then:

$$\begin{aligned}
 \text{BU}_s(x) &\triangleq \{x\} \\
 \text{BU}_s(x \circ i) &\triangleq \{x\} \\
 \text{BU}_s(p) &\triangleq \{p\} \\
 \text{BU}_s(P^{\leftarrow}) &\triangleq \{P^{\leftarrow}\} \\
 \text{BU}_s(P \circ Q) &\triangleq \{A \circ B \mid A \in \text{BU}_s(P) \wedge B \in \text{BU}_s(Q)\} \\
 \text{BU}_s(P \wedge Q) &\triangleq \text{BU}_s(P) \cup \text{BU}_s(Q)
 \end{aligned}$$

8.6.2 Weak beaming

When using the weak beam up and down operations, we do not consider the molecules of the hyperindex and hyperbase, but rather the presentations of these molecules. When beaming up or down, characterisations of the molecules must be derived from these presentations. The molecules from both the hyperindex and the hyperbase are characterised as follows:

$$\begin{aligned}
 \chi_{ind}(M) &\triangleq \{w \in \text{Names} \mid w \in n \wedge \rho(M, n, \alpha)\} \\
 \chi_{bse}(M) &\triangleq \{w \in \text{Names} \mid w \in n \wedge \rho(P, n, \alpha) \wedge P \in \text{BU}_s(M)\}
 \end{aligned}$$

As an example of characterisations based on the presentation of molecules, consider:

$$\begin{aligned}
 \chi_{ind}(\text{"president involved in marriage with politician"}) \\
 &= \{\text{involved in, marriage, politician, president, with}\} \\
 \chi_{ind}(\text{"president having been born in year"}) \\
 &= \{\text{born, having been, in, president, year}\}
 \end{aligned}$$

$$\begin{aligned}
 \chi_{bse}(\text{"president 'J.F. Kennedy' having been born in year 1917, and having died at age 46"}) \\
 &= \{\text{age, at, born, died, having, having been, in, president, year}\}
 \end{aligned}$$

where "president involved in marriage with politician" denotes the path expression (molecule) with verbalisation president involved in marriage with politician. Using these characterisations, we are able to relate molecules to each other to identify the relevance from one molecule to another. Several ways to determine the relevance between characterisations exist ([Rij75]). One possible, well known, way is:

$$\text{Rel}(C_1, C_2) \triangleq \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

For the example, we have the following relevance:

$$\begin{aligned}
 &\text{Rel}(\chi_{ind}(\text{"president having been born in year"}), \\
 &\quad \chi_{bse}(\text{"president 'J.F. Kennedy' having been born in year 1917, and having died at age 46"})) \\
 &= \frac{|\{\text{born, having been, in, president, year}\}|}{|\{\text{age, at, born, died, having, having been, in, president, year}\}|} \\
 &= \frac{5}{9}
 \end{aligned}$$

When carrying out a weak beam down from the hyperindex, the relevance between the molecules in the hyperbase, and the guide is determined. The results can be offered to the searchers, ordered by relevance, after which they can make a final selection. Conversely, when carrying out a weak beam up, the relevance between the molecules in the hyperindex and the current focus in the hyperbase can be determined. These results can also be offered to the user, ordered by relevance, after which they may decide upon a final destination.

8.7 Conclusions

In this chapter we investigated the integration between stratified hypermedia architecture and information systems, leading to a better and easier disclosure of the information stored in the information system. In the introduction to this chapter we argued that such a mechanism is required for meeting the first requirement on evolving information systems (subsection 1.3.1).

This integration enables the usage of query by navigation in EVORM disclosure schemas, given this integration, a user may interactively formulate a query. Using the query by navigation mechanism presented in this chapter, Elisa-D now also supports criterion L4 for query languages.

Chapter 9

Conclusions and Further Research

Maybe, just maybe, the dolphins were right after all, and it would have been better for man to have stayed in the oceans, and not have bothered to migrate to the land.

It certainly would have been a lot better for most other species on this beautiful, yet fragile, planet. The collective sanity of the human species, is indeed reflected by the very name the humans have given to this planet. What other species would refer to a planet who's surface is mostly covered by water as 'earth'? Every other intelligent species in the universe would have called this planet 'water'. The dolphins did.

From: "The Restaurant at the End of the Internet",
Erik Proper

9.1 Problems Statement Revisited

In this thesis we focussed on the development of a modelling technique, consisting of a way of working and a way of modelling, for evolving information systems. Due to the existence of the 'Methodology Jungle', we did not focus directly on a concrete (set of) modelling techniques, but rather tried first to develop a general theory. In this general theory, we tried to abstract as much as possible from intrinsic details of underlying modelling techniques.

In this section, we evaluate the contents of this thesis with respect to the requirements for evolving information systems, the technique independence of the general theory, and the general requirements on modelling techniques. These requirements were all discussed in chapter 1.

9.1.1 Support of evolution

We start by revisiting the set of requirements for evolving information systems presented in subsection 1.3.1, and evaluate the way of modelling (and communicating) for evolving information systems presented in this thesis.

1. *The system provides an adequate disclosure mechanism for the retrieval of all stored information.*

In section 6.3, the notion of disclosure schema was introduced, providing an information structure covering all stored information. The disclosure schema therefore guarantees complete disclosure. Further, Elisa-D (chapter 6) provides a disclosure language, allowing for the formulation of information requests in a naturalistic language. The ease of formulation in this language is even further enhanced by the stratified hypermedia approach provided in chapter 8.

As a result, an evolving information system built in conformance with the way of modelling and communicating of evolving information systems presented in this thesis, provides ample support for this requirement.

2. *The information system allows update of all information depending on the specific universe of discourse, maintaining a set of well-formedness rules.*

In section 7.4, we introduced Elisa-D transactions allowing for updates of all relevant parts of the disclosure schema, while maintaining any constraint present in the application model.

3. *The information system allows correction of all information (previously) recorded in the system.*

The update framework introduced in chapter 4, provides a correction mechanism for (evolving) information systems, nevertheless, in chapter 4, we have identified some problems related to the ability to correct. Decisions based on faulty information in the universe of discourse may already have been made, possibly leading to unrecoverable effects in the environment (organisation) of the information system.

4. *The system does not forget any information recorded in the information system unless explicitly told to.*

The way of modelling presented in this thesis clearly supports the first part of this requirement, since no information is forgotten. The second part, forgetting information when explicitly required, was not addressed in this thesis. It should, however, be obvious that in a real implementation of an EIMS such a facility must be present.

5. *Any update of the information system may not interrupt activities of the organisation.*

The presented way of modelling, does not endanger this requirement in any way. The extent to which an evolving information system is able to evolve without the need to interrupt activities of the organisation, depends mainly on the actual implementation.

In the development of evolving information systems, three objectives were distinguished. This thesis has only covered the first objective. The other two, implementation and a suitable way of working, remain issues for further research. In the next section, we discuss briefly some of the aspects involved in these two remaining objectives.

9.1.2 Technique independence

The general theory for the evolution of application models introduced in chapters 2 to 4 is applied to the PSM–LISA–D–Hydra framework in chapters 5 to 7, resulting in the EVORM–Elisa–D–Hydrae framework. This application proved that the general evolution theory made the development of a concrete modelling technique for evolving information systems easier. Further, a number of properties from the general theory turned out to be “theorems for free”.

As argued before, we have chosen the PSM–LISA–D–Hydra framework as a first application of the general evolution theory, as the underlying modelling techniques can be regarded as good representatives of an even wider range of modelling techniques.

The PSM–LISA–D–Hydra framework is clearly an integrated set of modelling techniques geared to the modelling of data intensive domains ([Hof93]). Ideally, the general theory should also be applied to modelling techniques for process intensive domains, and further to object oriented modelling techniques. As our primary aim was the development of a modelling technique for evolving information systems, these additional validations of the general theory remain issues for further research.

9.1.3 General requirements on modelling techniques

In subsection 1.2.1, we introduced five general requirements for information modelling techniques. These requirements covered: formal semantics, expressive power, conceptual level, comprehensibility and executability.

In [Hof93], it is already argued that the PSM–LISA–D–Hydra framework adheres to these requirements. Since the EVORM–Elisa–D–Hydrae framework has been developed out of the PSM–LISA–D–Hydra framework, the same motivations hold. The extension of the original modelling technique(s) by the notion of evolution, clearly does not endanger these requirements. Even more, by maintaining the evolution of elements, the expressiveness is extended. Consider, for example, the modelling of evolution of instances of power types as discussed in subsection 3.3.2.

9.2 Issues for Further Research

When implementing an evolving information management system, at least four problem areas are in need of further research. Problem area one is ‘forgetting’, evolving information systems need to be able to forget information when explicitly directed to do so. Problem area two is the internal representation of data stored in the evolving information system ([ISO87]). The third problem area is concerned with the fact that modern information systems tend to be distributed. Finally, a proper way of working for the maintenance of evolving information systems needs to be developed. These problem areas are discussed briefly in the remainder of this section.

9.2.1 Forgetting information

‘Forgetting’ a certain piece of information α from an application model H , leading to an application model H' , should at least adhere to the following rules:

1. No new information may be introduced: $H' \vdash \varphi \Rightarrow H \vdash \varphi$.
2. The forgotten information must indeed be forgotten: $H' \not\vdash \alpha$.

where $H \vdash \varphi$ means: φ can be derived (by a query) from application model history H , *without* applying the closed world assumption. These rules define the notion of forgetting information formally. Ideally, H' will be chosen such that $\{\varphi | H' \vdash \varphi\}$ is maximal, and H' still adheres to the above two rules. The ability to forget information has two motivations:

1. It is not viable to store the complete history of an evolving information system.
Compression techniques, as well as the gigantic storage capacity of the new generation of storage devices, may soften this problem. Nevertheless, obsolete information eventually needs to be forgotten, i.e. removed entirely from the data storage.
2. Further, the most important reason to be able to forget information, is the protection of privacy.
Privacy protection laws require that data about people should not be stored for longer than a given period of time. After this period of time, any information stored about persons must be removed (destroyed).

In the next two subsections, these two motivations are addressed in some more detail.

9.2.1.1 Obsolete information

In chapter 4, three levels of update were distinguished. Forgetting information, would lead to a fourth level of updates. The exact definition of the *forget* operator is highly dependent on the chosen modelling techniques for the application model. In particular on the ones chosen for modelling the information structure and the information base. Some major causes of concern when forgetting information, are:

1. *Scope of forgetting*
When removing obsolete information, one does not always simply want to remove all data older than a given age. Usually, one will be more selective, and determine an *obsolescence age* for each object type separately.
Further, one may want to forget detailed information about the past, but replace it with aggregated information. For instance, forgetting the day to day sales figures more than five years ago, and replacing them with the (derivable) yearly sales figures.
2. *Consistency of stored information*
When forgetting information about the past, an application model history is produced with ‘blank holes’ in it. These blank holes make it hard to enforce constraints on the history.

As an example, imagine a dynamic constraint referring to the past of instances of an object type (e.g. salaries may not decrease). When past information is forgotten, it may become impossible to enforce such dynamic constraints properly.

Ideally, when forgetting information the resulting application model history allows exactly the same set of updates. Formally:

Let H' be the application model resulting after a forget operation on H . Let m be a method, then we ideally have:

$$\forall_{t \in \mathcal{T}} [\exists_G [H \llbracket m \rrbracket_t G \wedge \text{IsAMH}(G)] \iff \exists_{G'} [H' \llbracket m \rrbracket_t G' \wedge \text{IsAMH}(G')]]$$

Note that when forgetting information this behaviour is generally not feasible.

9.2.1.2 Privacy and evolving information systems

To protect the privacy of people whose information has been recorded in the information system, it must be possible to remove that information. When forgetting the address and name of a former employee, one does not always want a reduction of the number of employees in the past. As a result, one cannot simply remove all traces from the former employee from memory.

A possible strategy to remove private information from an information system, without excluding the derivation of aggregated information, such as the total number of employees, is the *defoliation* of abstract instances. When forgetting all information about a former employee, it may suffice to destroy the identifications. These identifications may consist of address, name, social security number, etc. Information such as the salaries, the department to which the employee was assigned, however, may still be left intact. This strategy, raises the question of the compromisability of information systems ([Jon83], [SJR83], [MS90b]). One may be able to derive parts of the original information about the former employee, using information that remains in the information system.

9.2.2 Internal representation of the disclosure schema

In an evolving information system, the disclosure schema must be represented by internal structures. As the disclosure schema evolves with the course of time, these internal structures are also subject to evolution, and may therefore have to be changed dynamically with the course of time.

In general, the disclosure schema only increases with the course of time. The only exception being the forgetting of complete object types. Therefore, an evolutionary, or incremental, approach to the internal representation problem may be beneficial ([BW92a]).

9.2.3 Distributed evolving information systems

When using an (evolving) information system in a realistic environment, the need almost immediately arises to distribute the system. The distribution of the information system over multiple sites, which may even be on different continents, introduces a new set of problems. Some of these problems are independent of the evolution aspect of evolving information systems. Matters such as synchronisation, record locking, etc. can be dealt with in a similar way as that used in non-evolving information systems.

An evolving information system specific problem is, however, the distribution of the time axis. In a distributed environment, the time axis needs to be extended by the notion of location, leading to a distributed time axis:

$$\mathcal{T}_d \triangleq \mathcal{W} \times \mathcal{T}$$

where \mathcal{T} is a normal time axis, and \mathcal{W} is a set of locations (worlds). Application model histories can than be modelled as:

$$\mathcal{AMH}_d \triangleq \wp(\mathcal{T}_d \rightarrow \mathcal{AME})$$

An additional complication is the fact that there, of necessity, does not exist an increment operation on \mathcal{T}_d , but rather an increment relation. This makes it impossible to simply enforce the well-formedness rules from IsAMH on the resulting application model histories as a whole. Instead, the well-formedness rules from IsAMH should be enforced for each location $w \in \mathcal{W}$ separately. For every location, the ‘local’ application model history is still an element from \mathcal{AMH} . This discussion can be stated more formally as:

Let $H_d \in \mathcal{AMH}_d$, then:

$$\text{IsAMH}_d(H_d) \triangleq \forall_{w \in \mathcal{W}} [\text{IsAMH}(\sigma_w(H_d))]$$

where $\sigma_w(H_d) \triangleq \{\sigma_w(h_d) \mid h_d \in H_d\}$, and:

$$\sigma_w(h_d) \triangleq \lambda t.h(w, t)$$

Additionally, rules must be formulated enforcing global well-formedness and coherence between the evolutions on the separate locations. A first rule demanding such global coherence requires the equality of local element evolutions, disregarding delays:

For any $h \in \mathcal{AMH}_d$, and $w_1, w_2 \in \mathcal{W}$, there exists a function $glue : \mathcal{T} \rightarrow \mathcal{T}$ such that:

$$t_1 \leq t_2 \iff glue(t_1) \leq glue(t_2)$$

and:

$$\sigma_{w_1}(h) \cdot glue = \sigma_{w_2}(h)$$

Informally, this rule demands that element evolutions must behave similarly at locations, but they may take their time ($glue$) to do so.

9.2.4 Maintaining evolving information systems

This thesis is not concerned with a proper way of working, supporting the maintenance, and design, of an evolving information system. Such a way of working will incorporate rules for the detection of evolution steps in the universe of discourse. It is only obvious that such a way of working depends highly on the chosen modelling techniques of the application model. For the EVORM case, this means that the way of working will be related highly to the natural language approach taken by NIAM ([NH89]).

CASE tools are used to support information system development methods. In case of an EIMS, any CASE tool supporting the way of working and way of modelling will be an integral part of the management system. This integration is illustrated in figure 7.8. Whenever a change to the application model is needed, the current version of the application model history is transferred to the repository of the CASE tool. The needed modifications to the most recent application model are made using the CASE tool, and on completion communicated to the evolving information system.

The consistency of the changed application model will be checked by the CASE tool, before communicating the changes to the evolving information system. Further, the well-formedness of the evolution of the application model can be checked in advance, as the CASE tool has access to the entire application model history. To do this, the updated application model must be related to the application model history as a whole.

Appendix A

Mathematical Notations

The mathematical notation used in this thesis is described briefly in this appendix.

A.1 Sets

The power set of a set A , i.e. the set of all subsets of A , is denoted as $\mathcal{O}(A)$. The set of all finite sequences of elements from A is denoted by A^* , while the set of all *non-empty* finite sequences of elements from A is denoted by A^+ . The i -th element of a sequence $\langle a_1, \dots, a_i, \dots, a_n \rangle$, i.e. a_i , can be found by projection: $\langle a_1, \dots, a_i, \dots, a_n \rangle_{<i>}$. As well as the set operations: $\cup, \cap, \setminus, \subseteq$ with their usual meaning, we also define:

$$\begin{aligned} A \subset B &\triangleq A \subseteq B \wedge A \neq B \\ A \not\subset B &\triangleq \neg A \subset B \\ A \ominus x &\triangleq A \setminus \{x\} \\ X \oplus x &\triangleq A \cup \{x\} \end{aligned}$$

Some sets will have a total ordering, for a finite set X with a total ordering $< \subseteq X \times X$ we can define the extremes:

$$\begin{aligned} \max(X) &\triangleq x \in X \text{ such that } \forall_{a \in X} [a < x \vee a = x] \\ \min(X) &\triangleq x \in X \text{ such that } \forall_{a \in X} [x < a \vee a = x] \end{aligned}$$

A.2 Functions

A partial function f from A to B is defined by $f : A \rightarrow B$. Formally, it is a relation $f \subseteq A \times B$ such that $\langle a, b \rangle \in f \wedge \langle a, c \rangle \in f \Rightarrow b = c$. This property makes it possible to write $f(a) = b$ instead of $\langle a, b \rangle \in f$.

A function f is a set of binary tuples. The first and second values of these binary tuples are identified as:

$$\begin{aligned} \pi_1(f) &\triangleq \{a \mid \langle a, b \rangle \in f\} \\ \pi_2(f) &\triangleq \{b \mid \langle a, b \rangle \in f\} \end{aligned}$$

The following abbreviations are used for (partial) functions:

$$\begin{aligned} \text{dom}(f) &\triangleq \pi_1(f) \\ \text{ran}(f) &\triangleq \pi_2(f) \end{aligned}$$

$$\begin{aligned} f(a)\downarrow &\triangleq a \in \text{dom}(f) \\ f(a)\uparrow &\triangleq a \notin \text{dom}(f) \end{aligned}$$

For unary functions, we will write $f\downarrow a$, and $f\uparrow a$, instead of $f(a)\downarrow$, and $f(a)\uparrow$ respectively. Further, $f_1, \dots, f_n\downarrow a_1, \dots, a_m$ is employed as an abbreviation for: $\forall_{1 \leq i \leq n} \forall_{1 \leq j \leq m} [f_i\downarrow a_j]$.

A total function f from A to B is defined by $f : A \rightarrow B$. Formally, $f : A \twoheadrightarrow B$ for which $\text{dom}(f) = A$.

The (partial or total) function $f[A']$ is the function f restricted to a subdomain $A' \subseteq \text{dom}(f)$. This function is defined by:

$$f[A'] \triangleq \{ \langle a, b \rangle \in f \mid a \in A' \}$$

A notational shorthand for denoting tuples of functions is used in this thesis: $\langle p, a \rangle \triangleq p : a$. As an example consider a function f such that $f = \{ \langle p, a \rangle, \langle q, b \rangle \}$. This function will alternatively be denoted as $\{ p : a, q : b \}$. For a function f we define:

$$f \odot x : y \triangleq \{ v : w \in f \mid v \neq x \} \oplus x : y$$

In some cases, functions have to be denoted as an expression, for this purpose we apply lambda calculus ([Bar84]). An example is: $f = \lambda x. \text{if } x = 0 \text{ then } 0 \text{ else } 1/x \text{ fi}$. In the definition of functions, we will use \perp to indicate that the function is not defined for a particular value. Sometimes we will also use **error** instead of \perp , to indicate that the function/operation leads to an error situation.

The inverse of a function, or relation, R is defined as: $f^{\leftarrow} \triangleq \{ y : x \mid x : y \in f \}$. The inverse f^{\leftarrow} of a function, is a function if f is an injection (and thus a bijection on $\text{dom}(f)$). Functions f and g can be composed, if $\text{ran}(g) \subseteq \text{dom}(f)$ by \cdot as follows: $f \cdot g \triangleq \lambda x. f(g(x))$. Relations can be composed by \circ as: $R \circ S \triangleq \{ \langle x, z \rangle \mid \langle x, y \rangle \in R \wedge \langle y, z \rangle \in S \}$.

If h is a function $h : X \twoheadrightarrow Y$, and X is an ordered domain, then the last and first value to which a value has been assigned is identified by:

$$\begin{aligned} \text{Last}(h) &\triangleq \max \cdot \text{dom}(h) \\ \text{First}(h) &\triangleq \min \cdot \text{dom}(h) \end{aligned}$$

A.3 Proofs

In proofs, the \Rightarrow symbol stands for a logical deduction step, and a logical equivalence in a proof is denoted by \equiv . The motivation for a deduction step or equivalence is provided in parenthesis: $\{ \text{motivation of the step} \}$.

A.4 Tuples

Finally, for tuples over any domain, three operators exists:




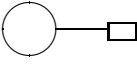
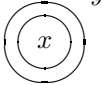
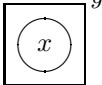
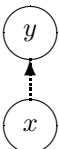
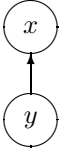

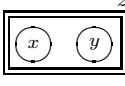





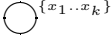
$$\begin{aligned} \text{Head}(\langle x_1, \dots, x_n \rangle) &\triangleq x_1 \\ \text{Tail}(\langle x_1, \dots, x_n \rangle) &\triangleq \langle x_2, \dots, x_n \rangle \\ \langle x_1, \dots, x_n \rangle ++ \langle y_1, \dots, y_n \rangle &\triangleq \langle x_1, \dots, x_n, y_1, \dots, y_n \rangle \end{aligned}$$

for these operators we have: $\langle \text{Head}(x) \rangle ++ \text{Tail}(x) = x$

Appendix B

EVORM Graphical Conventions

This appendix contains an overview of the EVORM symbols for object types, generalisations and specialisations, and graphical constraints used in this thesis.

object type:	
label type x :	
role:	
predicator:	
y power type of x :	
y sequence type of x :	
y is generalisation of x :	
y is specialisation of x :	
single-fact uniqueness constraint:	
schema type:	
uniqueness constraint:	
total role or cover constraint:	
exclusion constraint:	
subset constraint:	
equality constraint:	
enumeration constraint:	

Bibliography

- [AH87] S. Abiteboul and R. Hull. IFO: A Formal Semantic Database Model. *ACM Transactions on Database Systems*, 12(4):525–565, December 1987.
- [All84] J.F. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 1984(23):123–154, 1984.
- [Ari86] G. Ariav. A Temporally Oriented Data Model. *ACM Transactions on Database Systems*, 11(4):499–527, December 1986.
- [Ari91] G. Ariav. Temporally oriented data definitions: Managing schema evolution in temporally oriented databases. *Data & Knowledge Engineering*, 6(6):451–467, 1991.
- [ASU86] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, Massachusetts, 1986.
- [Avis95] D.E Avison. *Information Systems Development: Methodologies, Techniques and Tools*. McGraw-Hill, New York, New York, 2nd edition, 1995. ISBN 0077092333
- [Bar84] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, The Netherlands, Revised Edition, 1984.
- [Bas90] V.R. Basili. Viewing maintenance as reuse-oriented software development. *IEEE Software*, pages 19–25, January 1990.
- [BBB91] R. Bosman, R. Bouwman, and P.D. Bruza. The Effectiveness of Navigable Information Disclosure Systems. In G.A.M. Kempen, editor, *Proceedings of the Informatiewetenschap 1991 conference*, Nijmegen, The Netherlands, 1991.
- [BBMP95] G.H.W.M. Bronts, S.J. Brouwer, C.L.J. Martens, and H.A. Proper. A Unifying Object Role Modelling Approach. *Information Systems*, 20(3):213–235, 1995.
- [Bem87] Th.M.A. Bemelmans. *Bestuurlijke informatiesystemen en automatisering*. Stenfort Kroese, Leiden, The Netherlands, 3rd edition, 1987. In Dutch.
- [BF91] S. Brinkkemper and E.D. Falkenberg. Three Dichotomies in the Information System Methodology. In P.W.G. Bots, H.G. Sol, and I.G. Sprinkhuizen-Kuyper, editors, *Informatiesystemen in beweging*. Kluwer, Deventer, The Netherlands, 1991.
- [BHW91] P. van Bommel, A.H.M. ter Hofstede, and Th.P. van der Weide. Semantics and verification of object-role models. *Information Systems*, 16(5):471–495, October 1991.
- [Big88] J. Bigelow. Hypertext and CASE. *IEEE Software*, 5(2):23–27, 1988.
- [BKKK87] J. Banerjee, W. Kim, H.J. Kim, and H.F. Korth. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *SIGMOD Record*, 16(3):311–322, December 1987.
- [BMO⁺89] R. Bretl, D. Maier, A. Otis, D.J. Penney, B. Schuchardt, J. Stein, E.H. Williams, and M. Williams. The GemStone Data Management System. In W. Kim and F.H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, ACM Press, Frontier Series, pages 283–308. Addison-Wesley, Reading, Massachusetts, 1989.
- [Boi92] E.A. Boiten. *Views of Formal Program Development*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1992.
- [Bot89] P.W.G. Bots. *An Environment to Support Problem Solving*. PhD thesis, Delft University of Technology, Delft, The Netherlands, 1989.
- [BP88] B.W. Boehm and P.N. Papaccio. Understanding and controlling software costs. *IEEE Transactions of Software Engineering*, 14(10):1462–1477, October 1988.
- [BPW93] C.A.J. Burgers, H.A. Proper, and Th.P. van der Weide. Organising an Information System as Stratified Hypermedia. In H.A. Wijshoff, editor, *Proceedings of the Computing Science in the Netherlands Conference*, pages 109–120, Utrecht, The Netherlands, EU, November 1993.
- [Bri90] S. Brinkkemper. *Formalisation of Information Systems Modelling*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1990.
- [Bru93] P.D. Bruza. *Stratified Information Disclosure: A Synthesis between Information Retrieval and Hypermedia*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1993.
- [BS84] B.G. Buchanan and E.H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, Massachusetts, 1984.

- [Bub80] J.A. Bubenko. Information Modelling in the Context of System Development. In S.H. Lavington, editor, *Information Processing 80*, pages 395–411. North-Holland/IFIP, Amsterdam, The Netherlands, 1980.
- [Bub86] J.A. Bubenko. Information System Methodologies - A Research View. In T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors, *Information Systems Design Methodologies: Improving the Practice*, pages 289–318. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1986.
- [BW89] P. D. Bruza and Th.P. van der Weide. The semantics of data flow diagrams. In N. Prakash, editor, *Proceedings of the International Conference on Management of Data (CISMOD)*, pages 66–78, Hyderabad, India, 1989. McGraw-Hill Publishing Company.
- [BW90a] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge University Press, Cambridge, United Kingdom, 1990.
- [BW90b] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
- [BW90c] K.B. Bruce and P. Wegner. An algebraic model of subtype and inheritance. In F. Bancilhon and P. Buneman, editors, *Advances in Database Programming Languages*, ACM Press, Frontier Series, pages 75–96. Addison-Wesley, Reading, Massachusetts, 1990.
- [BW90d] P. D. Bruza and Th. P. van der Weide. Two level hypermedia - an improved architecture for hypertext. In A.M. Tjoa and R. Wagner, editors, *Proceedings of the Data Base and Expert System Applications Conference (DEXA 90)*, pages 76–83, Vienna, Austria, 1990. Springer-Verlag.
- [BW92a] P. van Bommel and Th.P. van der Weide. Reducing the search space for conceptual schema transformation. *Data & Knowledge Engineering*, 8:269–292, 1992.
- [BW92b] P.D. Bruza and Th.P. van der Weide. Stratified Hypermedia Structures for Information Disclosure. *The Computer Journal*, 35(3):208–220, 1992.
- [CH93] L.J. Campbell and T.A. Halpin. Automated Support for Conceptual to External Mapping. In S. Brinkkemper and F. Harmsen, editors, *Proceedings of the Fourth Workshop on the Next Generation of CASE Tools*, pages 35–51, Paris, France, June 1993.
- [Che76] P.P. Chen. The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.
- [Cho93] J. Chomicki. Temporal Databases. In *12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Washington D.C., May 1993. Tutorial Notes.
- [CR87] J. Clifford and A. Rao. A simple, general structure for Temporal Domains. In C. Rolland, F. Bodart, and M. Leonard, editors, *Temporal Aspects in information Systems*, pages 17–28. North-Holland/IFIP, Amsterdam, The Netherlands, 1987.
- [Cra86] T.C. Craven. *String Indexing*. Academic Press, London, United Kingdom, 1986.
- [Cus89] M.A. Cusumano. The software factory: A historical interpretation. *IEEE Software*, pages 23–30, March 1989.
- [CW83] J. Clifford and D.S. Warren. Formal Semantics for Time in Databases. *ACM Transactions on Database Systems*, 8(2):214–254, June 1983.
- [CW85] L. Cardelli and P. Wegner. On Understanding Types, Data Abstraction, and Polymorphism. *ACM Computing Surveys*, 17(4):471–522, December 1985.
- [CW93] M.A. Collignon and Th.P. van der Weide. An Information Analysis Method Based on PSM. In G.M. Nijssen, editor, *Proceedings of NIAM-ISDM. NIAM-GUIDE*, September 1993.
- [CY90] P. Coad and E. Yourdon. *Object-Oriented Analysis*. Yourdon Press, New York, New York, 1990.
- [Dav87] G.B. Davis. Strategies for Information Requirements Determination. In Robert Galliers, editor, *Information Analysis: Selected readings*, chapter 13. Addison-Wesley, Reading, Massachusetts, 1987.
- [Dav90] A.M. Davis. *Software Requirements: Analysis & Specification*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
- [DDP93] E. Dubois, P. Du Bois, and M. Petit. Eliciting and Formalising Requirements for C.I.M. Information Systems. In C. Rolland, F. Bodart, and C. Cauvet, editors, *Proceedings of the Fifth International Conference CAiSE'93 on Advanced Information Systems Engineering*, volume 685 of *Lecture Notes in Computer Science*, pages 252–274, Paris, France, 1993. Springer-Verlag.
- [De 88] O.M.F. De Troyer. On Rule-Based Generation of Conceptual Database Updates. In *Data and Knowledge*, pages 99–117, 1988.
- [De 91] O.M.F. De Troyer. The OO-Binary Relationship Model: A Truly Object Oriented Conceptual Model. In R. Andersen, J.A. Bubenko, and A. Sølvberg, editors, *Proceedings of the Third International Conference CAiSE'91 on Advanced Information Systems Engineering*, volume 498 of *Lecture Notes in Computer Science*, pages 561–578, Trondheim, Norway, May 1991. Springer-Verlag.
- [DHL⁺85] E. Dubois, J. Hagelstein, E. Lahou, A. Rifaut, and F. Williams. A Formalisation of Entities, Relationships, Attributes, and Events. Philips Manuscript M105, Philips Research Laboratory, Brussels, Belgium, 1985.
- [DMV88] O.M.F. De Troyer, R. Meersman, and P. Verlinden. RIDL* on the CRIS Case: A Workbench for NIAM. In T.W. Olle, A.A. Verrijn-Stuart, and L. Bhabuta, editors, *Information Systems Design Methodologies: Computerized Assistance during the Information Systems Life Cycle*, pages 375–459, Amsterdam, The Netherlands, EU, 1988. North-Holland/IFIP WG8.1.
- [EGH⁺92] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, and H-D. Ehrich. Conceptual modelling of database applications using an extended ER model. *Data & Knowledge Engineering*, 9(4):157–204, 1992.
- [EO93] C.A. Ewald and M.E. Orlowska. A Procedural Approach to Schema Evolution. In C. Rolland, F. Bodart, and C. Cauvet, editors, *Proceedings of the Fifth International Conference CAiSE'93 on Advanced Information Systems Engineering*, volume 685 of *Lecture Notes in Computer Science*, pages 22–38, Paris, France, 1993. Springer-Verlag.

- [EWH85] R. Elmasri, J. Weeldreyer, and A. Hevner. The category concept: An extension to the entity-relationship model. *Data & Knowledge Engineering*, 1:75–116, 1985.
- [FHL97] P.J.M. Frederiks, A.H.M. ter Hofstede, and E. Lippe. A Unifying Framework for Conceptual Data Modelling Concepts. *Information and Software Technology*, 39(1):15–25, January 1997.
- [FN85] A.L. Furtado and E.J. Neuhold. *Formal Techniques for Data Base Design*. Springer-Verlag, Berlin, Germany, 1985.
- [FOP92a] E.D. Falkenberg, J.L.H. Oei, and H.A. Proper. A Conceptual Framework for Evolving Information Systems. In H.G. Sol and R.L. Crosslin, editors, *Dynamic Modelling of Information Systems II*, pages 353–375. North-Holland, Amsterdam, The Netherlands, EU, 1992. ISBN 0444894055
- [FOP92b] E.D. Falkenberg, J.L.H. Oei, and H.A. Proper. A Metamodel for Update in Information Systems. Technical Report 92-05, Department of Information Systems, University of Nijmegen, Nijmegen, The Netherlands, EU, 1992.
- [FOP92c] E.D. Falkenberg, J.L.H. Oei, and H.A. Proper. Evolving Information Systems: Beyond Temporal Information Systems. In A.M. Tjoa and I. Ramos, editors, *Proceedings of the Data Base and Expert System Applications Conference (DEXA '92)*, pages 282–287, Valencia, Spain, EU, September 1992. Springer Verlag, Berlin, Germany, EU. ISBN 3211824006
- [FS76] J.P. Fry and E.H. Sibley. Evolution of Data-Base Management Systems. *Computing Surveys*, 8(1):7–42, 1976.
- [Gad88] S.K. Gadia. A Homogenous Relational Model and Query Language for Temporal Databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.
- [Gil88] T. Gilb. *Principles of software engineering management*. Addison-Wesley, Reading, Massachusetts, 1988.
- [GS90] P.K. Garg and W. Scacchi. A Hypertext System to Manage Software Life-Cycle Documents. *IEEE Software*, 7(3):90–98, 1990.
- [Hag92] T.M. Hagensen. Hyperstructure CASE Tools. In B. Theodoulidis and A. Sutcliffe, editors, *Proceedings of the Third Workshop on the Next Generation of CASE Tools*, pages 291–297, Manchester, United Kingdom, May 1992.
- [Hal89] T.A. Halpin. *A logical analysis of information systems: static aspects of the data-oriented perspective*. PhD thesis, University of Queensland, Brisbane, Australia, 1989.
- [Haw90] S. Hawking. The Arrow of Time. In *Yearbook of Science and the Future*, pages 44–53. Encyclopædia Britannica, 1990.
- [HE92] U. Hohenstein and G. Engels. SQL/EER-syntax and semantics of an entity-relationship-based query Language. *Information Systems*, 17(3):209–242, 1992.
- [HH93] T.A. Halpin and J. Harding. Automated Support for Verbalization of Conceptual Schemas. In S. Brinkkemper and F. Harmsen, editors, *Proceedings of the Fourth Workshop on the Next Generation of CASE Tools*, pages 151–161, Paris, France, June 1993.
- [HHO92] T.A. Halpin, J. Harding, and C-H. Oh. Automated Support for Subtyping. In B. Theodoulidis and A. Sutcliffe, editors, *Proceedings of the Third Workshop on the Next Generation of CASE Tools*, pages 99–113, Manchester, United Kingdom, May 1992.
- [HM81] M. Hammer and D. McLeod. Database Description with SDM: A Semantic Database Model. *ACM Transactions on Database Systems*, 6(3):351–386, September 1981.
- [HN93] A.H.M. ter Hofstede and E.R. Nieuwland. Task structure semantics through process algebra. *Software Engineering Journal*, 8(1):14–20, January 1993.
- [HNSE87] U. Hohenstein, L. Neugebauer, G. Saake, and H-D. Ehrich. Three-Level-Specification of Databases using an extended Entity-Relationship Model. In R.R. Wagner, R. Traunmüller, and H.C. Mayr, editors, *Informationsbedarfsermittlung und -analyse für den Entwurf von Informationssystemen*, pages 58–88, Berlin, Germany, 1987. Springer-Verlag.
- [HO92] T.A. Halpin and M.E. Orlowska. Fact-oriented modelling for data analysis. *Journal of Information Systems*, 2(2):97–119, April 1992.
- [Hof93] A.H.M. ter Hofstede. *Information Modelling in Data Intensive Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.
- [HPW92a] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. A Note on Schema Equivalence. Technical Report 92-30, Department of Information Systems, University of Nijmegen, Nijmegen, The Netherlands, EU, 1992.
- [HPW92b] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Data Modelling in Complex Application Domains. In P. Loucopoulos, editor, *Proceedings of the Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering*, volume 593 of *Lecture Notes in Computer Science*, pages 364–377, Manchester, United Kingdom, EU, May 1992. Springer Verlag, Berlin, Germany, EU. ISBN 3540554815
- [HPW92c] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Data Modelling in Complex Application Domains. In P. Loucopoulos, editor, *Proceedings of the Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering*, volume 593 of *Lecture Notes in Computer Science*, pages 364–377, Manchester, United Kingdom, EU, May 1992. Springer Verlag, Berlin, Germany, EU. ISBN 3540554815
- [HPW93] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, October 1993.
- [HPW94a] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. A Conceptual Language for the Description and Manipulation of Complex Information Models. In G. Gupta, editor, *Seventeenth Annual Computer Science Conference*, volume 16 of *Australian Computer Science Communications*, pages 157–167, Christchurch, New Zealand, January 1994. University of Canterbury. ISBN 047302313

- [HPW94b] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Supporting Information Disclosure in an Evolving Environment. In D. Karagiannis, editor, *Proceedings of the 5th International Conference DEXA'94 on Database and Expert Systems Applications*, volume 856 of *Lecture Notes in Computer Science*, pages 433–444, Athens, Greece, EU, September 1994. Springer Verlag, Berlin, Germany, EU. ISBN 3540584358
- [HRWL83] F. Hayes-Roth, D.A. Waterman, and D.B. Lenat. *Building Expert Systems*. Addison-Wesley, Reading, Massachusetts, 1983.
- [HSV89] K.M. van Hee, L.J. Somers, and M. Voorhoeve. Executable Specifications for Distributed Information Systems. In E.D. Falkenberg and P. Lindgreen, editors, *Information System Concepts: An In-depth Analysis*, pages 139–156. North-Holland/IFIP, Amsterdam, The Netherlands, 1989.
- [HV91] K.M. van Hee and P.A.C. Verkoulen. Integration of a data model and high-level petri nets. In *Proceedings of the Twelfth International Conference on Applications and Theory of Petri Nets*, pages 410–431, Gjern, Denmark, 1991.
- [HW92] A.H.M. ter Hofstede and Th.P. van der Weide. Formalisation of techniques: chopping down the methodology jungle. *Information and Software Technology*, 34(1):57–65, January 1992.
- [HW93] A.H.M. ter Hofstede and Th.P. van der Weide. Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10(1):65–100, February 1993.
- [ISO86] *Information Processing – Text and Office Systems – Standard General MarkUp Language (SGML)*, 1986. ISO 8879:1986. <http://www.iso.org>
- [ISO87] *Information processing systems – Concepts and Terminology for the Conceptual Schema and the Information Base*, 1987. ISO/TR 9007:1987. <http://www.iso.org>
- [ISO89] *Information technology – Open Document Architecture and interchange format: Introduction and general principles*. 1989. ISO/IEC 8613-1:1994. <http://www.iso.org>
- [JG87] D.A. Jardine and J.J. van Griethuysen. A logic-based information modelling language. *Data & Knowledge Engineering*, 2:59–81, 1987.
- [JMSV92] M. Jarke, J. Mylopoulos, J.W. Schmidt, and Y. Vassiliou. DAIDA: An Environment for Evolving Information Systems. *ACM Transactions on Information Systems*, 20(1):1–50, January 1992.
- [Jon83] W. de Jonge. Compromising Statistical Databases Responding to Queries about Means. *ACM Transactions on Database Systems*, 8(1):60–80, 1983.
- [Kat90] R.H. Katz. Toward a Unified Framework for Version Modelling in Engineering Databases. *ACM Computing Surveys*, 22(4):375–408, 1990.
- [KBC⁺89] W. Kim, N. Ballou, H.-T. Chou, J.F. Garza, and D. Woelk. Features of the ORION Object-Oriented Database. In W. Kim and F.H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, ACM Press, Frontier Series, pages 251–282. Addison-Wesley, Reading, Massachusetts, 1989.
- [Ken84] F. Kensing. Towards Evaluation of Methods for Property Determination: A Framework and a Critique of the Yourdon-DeMarco Approach. In Th.M.A. Bemelmans, editor, *Beyond Productivity: Information Systems Development for Organizational Effectiveness*, pages 325–338. North-Holland, Amsterdam, The Netherlands, 1984.
- [KM90] J. Kramer and J. Magee. The Evolving Philosophers Problem: Dynamic Change Management. *IEEE Transactions on Software Engineering*, 16(11):1293–1306, November 1990.
- [KW92] K. Kumar and R.J. Welke. Method Engineering: A Proposal for Situation-Specific Methodology Construction. In W.W. Cotterman and J.A. Senn, editors, *Systems Analysis and Design: A Research Agenda*, pages 257–269. Wiley, New York, New York, 1992.
- [Lam91] L. Lamport. The Temporal Logic of Actions. Report 79, Digital, Systems Research Center, Palo Alto, California, December 1991.
- [Lan87] F. Land. Adapting to Changing User Requirements. In Robert Galliers, editor, *Information Analysis: Selected readings*, chapter 12. Addison-Wesley, Reading, Massachusetts, 1987.
- [Lev79] A. Levy. *Basic Set Theory*. Springer-Verlag, Berlin, Germany, 1979.
- [Lew85] A. Lew. *Computer Science: A Mathematical Introduction*. Prentice-Hall, Englewood Cliffs, New Jersey, 1985.
- [LG91] P.J.F. Lucas and L.C. van der Gaag. *Principles of Expert Systems*. Addison-Wesley, Reading, Massachusetts, 1991.
- [LGN81] M. Lundeberg, G. Goldkuhl, and A. Nilsson. *Information Systems Development - A Systematic Approach*. Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
- [Lip92] E. Lippe. *Camera: Support for Distributed Cooperative Work*. PhD thesis, University of Utrecht, Utrecht, The Netherlands, 1992.
- [LS80] B. Lientz and E. Swanson. *Software Maintenance Management – a study of the maintenance of computer application software in 487 data processing organizations*. Addison-Wesley, Reading, Massachusetts, 1980. ISBN 0201042053
- [LS87] U.W. Lipeck and G. Saake. Monitoring dynamic integrity constraints based on temporal logic. *Information Systems*, 12(3):255–269, 1987.
- [Mai88] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Rockville, Maryland, 1988.
- [Mar77] M.E. Maron. On Indexing, Retrieval and the Meaning of About. *Journal of the American Society for Information Science*, 28(1):38–43, 1977.

- [Mat81] L. Mathiassen. *Systemudvikling og Systemudviklings-Metode*. PhD thesis, Aarhus University, Aarhus, Denmark, 1981. (In Danish).
- [MBJK90] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing Knowledge about Information Systems. *ACM Transactions on Information Systems*, 8(4):325–362, 1990.
- [McC89] C.L. McClure. *CASE is Software Automation*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989. ISBN 0131193309
- [McK86] E. McKenzie. Bibliography: Temporal Databases. *SIGMOD Record*, 15(4):40–52, December 1986.
- [Mey88] B. Meyer. *Object-oriented software construction*. Prentice-Hall, Englewood Cliffs, New Jersey, 1988.
- [MS90a] E. McKenzie and R. Snodgrass. Schema evolution and the relational algebra. *Information Systems*, 15(2):207–232, 1990.
- [MS90b] M. Miller and J. Seberry. Audit Expert and Statistical Database Security. In *Proceedings of the Australian Database Research Conference*, Databases in the 1990s, pages 149–174, Melbourne, Australia, February 1990. World Scientific.
- [MSW92] P. McBrien, A.H. Seltviet, and B. Wangler. An Entity-Relationship Model Extended to describe Historical Information. In A.K. Majumdar and N. Prakash, editors, *Proceedings of the International Conference on Information Systems and Management of Data (CISMOD 92)*, pages 244–260, Bangalore, India, July 1992.
- [NA87] S.B. Navathe and R. Ahmed. TSQL-A Language Interface for History Databases. In C. Rolland, F. Bodart, and M. Leonard, editors, *Temporal Aspects in Information Systems*, pages 113–128. North-Holland/IFIP, Amsterdam, The Netherlands, 1987.
- [NH89] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach*. Prentice-Hall, Sydney, Australia, 1989. ASIN 0131672630
- [NR89] G.T. Nguyen and D. Rieu. Schema evolution in object-oriented database systems. *Data & Knowledge Engineering*, 4:43–67, 1989.
- [OHM⁺88] T.W. Olle, J. Hagelstein, I.G. Macdonald, C. Rolland, H.G. Sol, F.J.M. van Assche, and A.A. Verrijn-Stuart. *Information Systems Methodologies: A Framework for Understanding*. Addison-Wesley, Reading, Massachusetts, USA, 1988. ISBN 0-201-54443-1
- [Oho90] A. Ohori. Orderings and Types in Databases. In F. Bancilhon and P. Buneman, editors, *Advances in Database Programming Languages*, ACM Press, Frontier Series, pages 97–116. Addison-Wesley, Reading, Massachusetts, 1990.
- [OPF92] J.L.H. Oei, H.A. Proper, and E.D. Falkenberg. Modelling the Evolution of Information Systems. Technical Report 92-36, Department of Information Systems, University of Nijmegen, Nijmegen, The Netherlands, EU, 1992.
- [OPF94] J.L.H. Oei, H.A. Proper, and E.D. Falkenberg. Evolving Information Systems: Meeting the Ever-Changing Environment. *Information Systems Journal*, 4(3):213–233, 1994.
- [Par90] H. Partsch. *Specification and Transformation of Programs - a Formal Approach to Software Development*. Springer-Verlag, Berlin, Germany, 1990.
- [PL93] P. Poncelet and L. Lakhal. Consistent Structural Updates for Object Database Design. In C. Rolland, F. Bodart, and C. Cauvet, editors, *Proceedings of the Fifth International Conference CAISE'93 on Advanced Information Systems Engineering*, volume 685 of *Lecture Notes in Computer Science*, pages 1–21, Paris, France, 1993. Springer-Verlag.
- [Pro93] H.A. Proper. Towards an Integration of Evolving Information Systems and CASE-Tools. In S. Brinkkemper and F. Harmsen, editors, *Proceedings of the Fourth Workshop on the Next Generation of CASE Tools*, pages 23–33, Paris, France, EU, June 1993. ISSN 09243755
- [PS87] D.J. Penney and J. Stein. Class Modification in the GemStone Object-Oriented DBMS. In N. Meyrowitz, editor, *Proceedings of the ACM Conference of Object-Oriented Systems, Languages and Applications (OOPSLA)*, pages 111–117, Orlando, Florida, October 1987.
- [PW93] H.A. Proper and Th.P. van der Weide. Towards a General Theory for the Evolution of Application Models. In M.E. Orłowska and M.P. Papazoglou, editors, *Proceedings of the Fourth Australian Database Conference*, Advances in Database Research, pages 346–362, Brisbane, Australia, February 1993. World Scientific, Singapore. ISBN 981021331X
- [PW94] H.A. Proper and Th.P. van der Weide. EVORM - A Conceptual Modelling Technique for Evolving Application Domains. *Data & Knowledge Engineering*, 12:313–359, 1994.
- [PW95a] H.A. Proper and Th.P. van der Weide. A General Theory for the Evolution of Application Models. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):984–996, December 1995.
- [PW95b] H.A. Proper and Th.P. van der Weide. Information Disclosure in Evolving Information Systems: Taking a shot at a moving target. *Data & Knowledge Engineering*, 15:135–168, 1995.
- [Ram94] G.J. Ramackers. *Integrated Object Modelling, an Executable Specification Framework for Business Analysis and Information System Design*. PhD thesis, University of Leiden, Leiden, The Netherlands, 1994.
- [RBP⁺91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.
- [Rij75] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, United Kingdom, 1975.
- [Rod91] J.F. Roddick. Dynamically changing schemas within database models. *The Australian Computer Journal*, 23(3):105–109, August 1991.
- [RP92] J.F. Roddick and J.D. Patrick. Temporal semantics in information systems - A survey. *Information Systems*, 17(3):249–267, 1992.
- [RV90] G.J. Ramackers and A.A. Verrijn-Stuart. First and Second order Dynamics in Information Systems. In H.G. Sol and K.M. van Hee, editors, *Proceedings of the International Working Conference on Dynamic Modelling of Information Systems*, Noordwijkerhout, The Netherlands, April 1990.

- [SA85] R. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 236–246, Austin, Texas, 1985.
- [SA86] R. Snodgrass and I. Ahn. Temporal Databases. *IEEE Computer*, 19(9):35–42, 1986.
- [Saa88] G. Saake. *Spezifikation, Semantik und Überwachung von Objekt-Lebensläufen in Datenbanken*. PhD thesis, Technische Universität Braunschweig, Braunschweig, Germany, 1988. (In German).
- [Saa91] G. Saake. Descriptive Specification of Database Object Behaviour. *Data & Knowledge Engineering*, 6(1):47–73, 1991.
- [Sal89] G. Salton. Automatic Indexing. In *Automatic Text Processing—The Transformation, Analysis, and Retrieval of Information by Computer*, chapter 9. Addison-Wesley, Reading, Massachusetts, 1989.
- [Sch84] G. Scheschonk. *Eine auf Petri-Netzen basier-en-de Konstruk-tion-s, Ana-ly-se und (Teil)Veri-fica-tion-s-me-tho-de zur Model-lierungsunterstützung bei der Entwicklung von Informationssystemen*. PhD thesis, Berlin University of Technology, Berlin, Germany, 1984. (In German).
- [SDBW91] P.L. van der Spiegel, J.T.W. Driessen, P.D. Bruza, and Th.P. van der Weide. A Transaction Model for Hypertext. In D. Karagiannis, editor, *Proceedings of the Data Base and Expert System Applications Conference (DEXA 91)*, pages 281–286, Berlin, Germany, 1991. Springer-Verlag.
- [Shi81] D.W. Shipman. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, March 1981.
- [SJR83] G.L. Sicherman, W. de Jonge, and R.P. van der Riet. Answering Queries Without Revealing Secrets. *ACM Transactions on Database Systems*, 8(1):41–59, March 1983.
- [SM83] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill New York, NY, 1983.
- [SM88] S. Shlaer and S.J. Mellor. *Object-Oriented Systems Analysis: Modeling the World in Data*. Yourdon Press, New York, New York, 1988.
- [Sno87] R. Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.
- [Sno90] R. Snodgrass. Temporal Databases Status and Research Directions. *SIGMOD Record*, 19(4):83–89, December 1990.
- [Sol83] H.G. Sol. A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations. In T.W. Olle, H.G. Sol, and C.J. Tully, editors, *Information Systems Design Methodologies: A Feature Analysis*, pages 1–7. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1983. ISBN 0-444-86705-8.
- [Sol84] H.G. Sol. Prototyping: A Methodological Assessment. In R. Budde et al., editors, *Approaches to Prototyping*. Springer-Verlag, Berlin, Germany, 1984.
- [Sol88] H.G. Sol. Information Systems Development: A Problem Solving Approach. In *Proceedings of 1988 INTEC Symposium Systems Analysis and Design: A Research Strategy*, Atlanta, Georgia, 1988.
- [Som89] I. Sommerville. *Software Engineering*. Addison-Wesley, Reading, Massachusetts, USA, 1989.
- [Sto77] J.E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Semantics*. MIT Press, Cambridge, Massachusetts, 1977.
- [Stu92] W.J.J. Stut jr. *Constructing Large Conceptual Models with MOVIE*. PhD thesis, University of Leiden, Leiden, The Netherlands, 1992.
- [SWS89] P.S. Seligmann, G.M. Wijers, and H.G. Sol. Analyzing the structure of I.S. methodologies, an alternative approach. In R. Maes, editor, *Proceedings of the First Dutch Conference on Information Systems*, Amersfoort, The Netherlands, EU, 1989.
- [SZ86] A.H. Skarra and S.B. Zdonik. The Management of Changing Types in an Object-Oriented Database. In N. Meyrowitz, editor, *Proceedings of the ACM Conference of Object-Oriented Systems, Languages and Applications (OOPSLA)*, pages 483–495, Portland, Oregon, September 1986.
- [Tan86] A.U. Tansel. Adding time dimension to relational model and extending relational algebra. *Information Systems*, 11(4):343–355, 1986.
- [TC90] A. Tuzhilin and J. Clifford. A Temporal Relational Algebra as a Basis for Temporal Relational Completeness. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proceedings of the 16th VLDB Conference*, pages 13–23, Brisbane, Australia, 1990.
- [TLW91] C. Theodoulidis, P. Loucopoulos, and B. Wangler. A conceptual modelling formalism for temporal database applications. *Information Systems*, 16(4):401–416, 1991.
- [TP89] T.H. Tse and L. Pong. Towards a Formal Definition for DeMarco Data Flow Diagrams. *The Computer Journal*, 32(1):1–12, 1989.
- [Tre91] M.T. Tresch. A Framework for Schema Evolution by Meta Object Manipulation. In *Proceedings of the 3d International Workshop on Foundations of Models and Languages for Data and Objects*, Aigen, Austria, September 1991. Institut für Informatik, TU Clausthal.
- [TS92] M.T. Tresch and M.H. Scholl. Meta Object Management and its Application to Database Evolution. In G. Pernul and A.M. Tjoa, editors, *11th International Conference on the Entity-Relationship Approach*, volume 645 of *Lecture Notes in Computer Science*, pages 299–321, Karlsruhe, Germany, October 1992. Springer-Verlag.
- [Ull89] J.D. Ullman. *Principles of Database and Knowledgebase Systems*, volume I, chapter 3. Computer Science Press, Rockville, Maryland, 1989.

- [Ver89] A.A. Verrijn-Stuart. Some Reflections on the Namur Conference on Information Systems Concepts. In E.D. Falkenberg and P. Lindgreen, editors, *Information System Concepts: An In-depth Analysis*. North-Holland/IFIP, Amsterdam, The Netherlands, 1989.
- [Ver93a] T.F. Verhoef. *Effective Information Modelling Support*. PhD thesis, Delft University of Technology, Delft, The Netherlands, EU, 1993. ISBN 9090061762
- [Ver93b] P.A.C. Verkoulen. *Integrated Information System Design - An Approach Based on Object-Oriented Concepts and Petri Nets*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 1993.
- [VW91] Th.H. Visschedijk and R.N. van der Werff. (R)evolutionary system development in practice. *Journal of Software Research*, pages 46–57, December 1991. Special Issue.
- [Wat91] D.A. Watt. *Programming Language Syntax and Semantics*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.
- [WBGW73] S.E. Willner, A.E. Bandurski, W.C. Gorhan, and M.A. Wallace. COMRADE data management system. In *Proceedings of the AFIPS National Computer Conference*, pages 339–345, Montvale, New Jersey, 1973. AFIPS Press.
- [Wel88] R.J. Welke. The CASE Repository: More than another database application. In W.W. Cotterman and J.A. Senn, editors, *Proceedings of 1988 INTEC Symposium Systems Analysis and Design: A Research Strategy*, Atlanta, Georgia, 1988. Georgia State University.
- [WH90] G.M. Wijers and H. Heijes. Automated Support of the Modelling Process: A view based on experiments with expert information engineers. In B. Steinholz, A. Solvberg, and L. Bergman, editors, *Proceedings of the Second Nordic Conference CAiSE'90 on Advanced Information Systems Engineering*, volume 436 of *Lecture Notes in Computer Science*, pages 88–108, Stockholm, Sweden, EU, 1990. Springer-Verlag. ISBN 3540526250
- [WHO92] G.M. Wijers, A.H.M. ter Hofstede, and N.E. van Oosterom. Representation of Information Modelling Knowledge. In V.-P. Tahvanainen and K. Lyytinen, editors, *Next Generation CASE Tools*, volume 3 of *Studies in Computer and Communication Systems*, pages 167–223. IOS Press, 1992.
- [Wig90] R.E. Wiggins. Document Image Processing - New Light on an Old Problem. *International Journal of Information Management*, 10(4):297–318, 1990.
- [Wij91] G.M. Wijers. *Modelling Support in Information Systems Development*. PhD thesis, Delft University of Technology, Delft, The Netherlands, 1991. ISBN 9051701101
- [Win90] J.J.V.R. Wintraecken. *The NIAM Information Analysis Method: Theory and Practice*. Kluwer, Deventer, The Netherlands, EU, 1990.
- [WJL91] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with the Granularity of Time in Temporal Databases. In R. Andersen, J.A. Bubenko, and A. Sølvyberg, editors, *Proceedings of the Third International Conference CAiSE'91 on Advanced Information Systems Engineering*, volume 498 of *Lecture Notes in Computer Science*, pages 124–140, Trondheim, Norway, May 1991. Springer-Verlag.
- [WMP⁺76] A. van Wijngaarden, B.J. Mailloux, J.E.L. Peck, C.H.A. Koster, M. Sintzoff, C.H. Lindsey, L.T. Meertens, and R.G. Fisker. *Revised Report on the Algorithmic Language ALGOL 68*. Springer-Verlag, Berlin, Germany, 1976.
- [WW88] Y. Wand and R. Weber. An Ontological Analysis of some fundamental Information Systems Concepts. In *Proceedings of the Ninth International Conference on Information Systems*, pages 213–226, Minesota, Mineapolis, November 1988.
- [You89] E. Yourdon. *Modern Structured Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

Samenvatting

Teneinde hun overlevingskansen te vergroten, dienen organisaties zich aan te kunnen passen aan hun steeds sneller veranderende omgeving. Veranderingen die bijvoorbeeld worden teweeggebracht door de Europese eenwording, de invoering van nieuwe technologieën en de constant veranderende (belasting)wetgevingen. Als het primaire proces van een organisatie verandert zal ook de informatiebehoefte van deze organisatie veranderen. Dit heeft tot gevolg dat in gebruik zijnde informatiesystemen eveneens aangepast moeten worden. Deze veranderingen dienen zoveel mogelijk on-line te geschieden, dat wil zeggen dat het informatiesysteem slechts gedurende korte tijd niet ter beschikking van de gebruikers mag staan.

Momenteel wordt door het gebruik van CASE-tools en objectgeoriënteerde benaderingen gepoogd de doorlooptijd van de zogenaamde software-ontwikkelingscyclus te verkorten. Een blijvend probleem bij deze benaderingen is echter dat veranderingen van structurele aard niet on-line kunnen gebeuren en dat de informatie die in oude informatiesystemen opgeslagen ligt vaak (deels) verloren gaat. In dit proefschrift wordt daarom een nieuwe klasse van informatiesystemen ingevoerd, de *evoluerende informatiesystemen*.

In hoofdstuk 1 worden de eisen die wij aan evoluerende informatiesystemen stellen besproken. Een evoluerend informatiesysteem heeft als belangrijke kenmerken dat alle applicatie-afhankelijke aspecten veranderbaar zijn en dat een historie (de evolutie) van alle veranderingen wordt bijgehouden. Daarnaast moet een verandering in het evoluerende informatiesysteem on-line kunnen gebeuren. De applicatie-afhankelijke aspecten betreffen niet alleen de opgeslagen informatie, maar ook de structuur van de opgeslagen informatie en acties die daar eventueel op uitgevoerd kunnen worden. In dit proefschrift wordt de term *applicatiemodel* ingevoerd voor het complete model van een universum van discussie (de applicatie), dat zowel de structurele aspecten van de opgeslagen informatie als de opgeslagen informatie zelf (de populatie) bevat. In een evoluerend informatiesysteem kan het gehele applicatiemodel aan evolutie onderhevig zijn.

Bij het ontwikkelen van een evoluerend informatiesysteem onderscheiden we drie hoofddoelen. Ten eerste moet er een gegevensmodelleringstechniek ontwikkeld worden die evolutie van informatiesystemen ondersteunt. Gegeven zo'n techniek moet er vervolgens een implementatie gemaakt worden van een EIMS (Evolving Information System Management System). Vervolgens moet er een werkwijze ontwikkeld worden om op adequate wijze om te gaan met evolutie van de informatiebehoeften in/van organisaties en die in het informatiesysteem te modelleren (weer te geven).

Dit proefschrift concentreert zich op het eerste doel, de definitie van een gegevensmodelleringstechniek die het modelleren van evolutie ondersteunt. Aan gegevensmodelleringstechnieken in het algemeen kunnen een aantal belangrijke eisen gesteld worden.

Bij het vervullen van het eerste doel kan het beste uitgegaan worden van een bestaande modelleringstechniek voor informatiesystemen, om te voorkomen dat opnieuw het wiel wordt uitgevonden. Er bestaan echter vele gegevensmodelleringstechnieken en het is derhalve niet verstandig om op voorhand een concrete modelleringstechniek te kiezen. In dit proefschrift wordt daarom eerst een algemene benadering voor het definiëren van zo'n modelleringstechniek uitgezet. Aan de hand van deze algemene benadering wordt vervolgens een algemene evolutietheorie ontwikkeld. Deze theorie wordt daarna op een aantal concrete modelleringstechnieken toegepast (het PSM-LISA-D-Hydra-raamwerk). Die toepassing resulteert in evolutie ondersteunende modelleringstechnieken (het EVORM-Elisa-D-Hydrae-raamwerk).

In hoofdstuk 2 wordt het *informatiestructuur-universum* ingevoerd, dat een aantal gemeenschappelijke kenmerken van gegevensmodelleringstechnieken in kaart brengt. Hierbij staan vier eigenschappen centraal. Ten eerste wordt er onderscheid gemaakt tussen objecttypen waarvan instanties wel of niet direct communiceerbaar zijn. Dit onderscheid resulteert in een concrete en een abstracte wereld. Verder kan de instantiëring van twee objecttypen waarvan gemeenschappelijk hebben. Zulke objecttypen worden dan *typegerelateerd* genoemd. Objecttypen kunnen ook eigenschappen van elkaar erven omdat zij in termen van elkaar gedefinieerd kunnen zijn. Dit resulteert in de zogenaamde *identificatie-hiërarchie*. Ten slotte wordt er een kenmerk ingevoerd om aan te kunnen geven of een gegeven groep van objecttypen tezamen een correcte informatiestructuur vormt.

De evolutie van een informatiestructuur in een informatiestructuur-universum en de daarin herkende kenmerken dienen zich volgens een aantal wetten te gedragen. Met behulp van deze wetten kunnen al een aantal krachtige stellingen bewezen worden die verderop in het proefschrift, bij de toepassing op een concrete modelleringstechniek, tot bruikbare eigenschappen van de elementen in de applicatiemodellen leiden.

Hoofdstuk 3 begint met het bespreken van een aantal wijzen waarop men de evolutie van een applicatiemodel kan modelleren. Vervolgens wordt het concept *applicatiemodel-universum* ingevoerd als overkoepelend concept voor versies van applicatiemodellen. Naast het eerder ingevoerde informatiestructuur-universum bevat het applicatiemodel-universum de andere mogelijke elementen van een applicatiemodel, te weten de constraints, de populatie en het activiteitenmodel.

Binnen een applicatiemodel-universum kunnen versies van applicatiemodellen worden onderscheiden. Voor dergelijke versies worden welgevormdheidseisen geformuleerd. Tevens worden er welgevormdheidseisen geformuleerd voor veranderingen van applicatiemodelversies. Met deze laatste welgevormdheidseisen wordt voorkomen dat applicatiemodellen al te “wild” kunnen evolueren.

In een evoluerend informatiesysteem wordt de gehele historie van de elementen, die tezamen het applicatiemodel vormen, bijgehouden. Een direct gevolg hiervan is dat de bekende scheiding tussen *opslagtijd* (de tijd waarop veranderingen in het informatiesysteem plaats vinden) en *werkelijke tijd* (de tijd waarop de veranderingen in het universum van discussie plaats vinden) ook in een evoluerend informatiesysteem aanwezig dient te zijn. Daarnaast moet een evoluerend informatiesysteem ook om kunnen gaan met correcties van reeds verwerkte veranderingen. Er kunnen immers altijd fouten gemaakt worden bij het doorvoeren van veranderingen in het informatiesysteem. In hoofdstuk 4 worden daarom drie niveaus van algebra’s ingevoerd, die de drie abstractieniveaus (werkelijke tijd, opslagtijd en correctie) formeel invoeren en onderling relateren.

De hoofdstukken 2 tot en met 4 hebben de algemene evolutietheorie ingevoerd. In de hoofdstukken 5 tot en met 7 wordt die theorie op een geïntegreerd stel modelleringstechnieken, het reeds eerder genoemde PSM–LISA–D–Hydra-raamwerk, toegepast.

In hoofdstuk 5 wordt de theorie toegepast op de informatiemodelleringstechniek PSM. Deze toepassing resulteert in de evolutie ondersteunende gegevensmodelleringstechniek EVORM en bestaat uit vier hoofdstappen. Als eerste wordt het EVORM informatiestructuur-universum gedefinieerd. Als tweede wordt bewezen dat dit universum een correct informatiestructuur-universum in de zin van de algemene evolutietheorie is. In stap drie worden de welgevormdheidseisen voor versies en evoluties van EVORM-gegevensmodellen met een aantal EVORM-specifieke eisen uitgebreid. Laatstelijk worden de eigenschappen die voor de algemene theorie zijn bewezen geïnterpreteerd voor EVORM.

In hoofdstuk 6 wordt de bestaande opvraagtaal LISA-D uitgebreid tot een taal (Elisa-D) die evoluerende gegevensmodellen ondersteunt. LISA-D is een opvraagtaal waarin de vraagstellingen een semi-natuurlijke taal formaat hebben. Dit formaat wordt voor een belangrijk deel bereikt door betekenisvolle naamgeving, die als gevolg van de natuurlijke-taal benadering van NIAM gebruikt wordt in de EVORM schema’s. Alvorens Elisa-D in te voeren, wordt een semantisch domein ingevoerd waarin uiteindelijk de semantiek van Elisa-D uitgedrukt kan worden. Dit domein bestaat uit de zogenaamde *padexpressies*, die op hun beurt zijn gedefinieerd in termen van een gegeneraliseerde klasse van *multisets*.

Omdat alle aspecten van het applicatiemodel, inclusief de historie, opvraagbaar moeten zijn, wordt het zogenaamde *ontsluitingsschema* ingevoerd. Dit schema bevat alle versies van de informatiestructuur uit de historie van applicatiemodellen en de *meta-modellen* van de gebruikte modelleringstechnieken. Het zo verkregen ontsluitingsschema

is de informatiestructuur van alle informatie die is opgeslagen in het informatiesysteem. Elisa-D gebruikt dan ook dit laatste schema voor het formuleren van vraagstellingen. Naast de formulering van vraagstellingen kan Elisa-D ook gebruikt worden voor het formuleren van constraints. De expressiviteit van Elisa-D kan groot genoemd worden, daar Elisa-D onder andere een macromechanisme met “lazy evaluation” biedt waarmee recursieve vraagstellingen geformuleerd kunnen worden.

In hoofdstuk 7 wordt het laatste deel van het applicatiemodel afgedekt, door het toepassen van de evolutietheorie op de activiteitenmodelleringstechniek Hydra. Deze laatste toepassing resulteert in Hydrae. Daar deze toepassing geen invloed heeft op de eigenlijke semantiek van Hydra-modellen, wordt in dit hoofdstuk in eerste instantie slechts de syntax van Hydrae-modellen beschreven. De kern van de integratie tussen EVORM en Hydrae wordt gevormd door zogenaamde transacties. Deze transacties worden in dit hoofdstuk ingevoerd als een uitbreiding van Elisa-D.

Een belangrijke eis, zoals die in hoofdstuk 1 geformuleerd zal worden, voor evoluerende informatiesystemen en informatiesystemen in het algemeen, is dat er een goed informatie-ontsluitingsmechanisme aanwezig is. Elisa-D is een eerste stap in deze richting, maar zodra het ontsluitingsschema groot wordt (honderden objecttypen zijn niet uit te sluiten), kan de gebruiker alsnog het overzicht kwijt raken. In dit hoofdstuk wordt een, door de hypermedia-wereld geïnspireerd, ontsluitingsmechanisme aangedragen waarin Elisa-D queries worden geformuleerd door het computer ondersteund “navigeren” door de informatiestructuur.

In hoofdstuk 9 wordt tenslotte de eindafrekening gepresenteerd. Het EVORM–Elisa-D–Hydrae raamwerk wordt geëvalueerd in het kader van de eisen voor een evoluerend informatiesysteem en gegevensmodelleringstechnieken in het algemeen, zoals gesteld in hoofdstuk 1. Het zal blijken dat dit raamwerk in principe voldoet aan de gestelde eisen, al is verder onderzoek noodzakelijk om te komen tot een daadwerkelijke implementatie van een EIMS. Tenslotte worden er een aantal belangrijke aandachtsgebieden besproken voor verder onderzoek.

Curriculum Vitae

De auteur van dit proefschrift werd op 22 mei 1967 geboren te Rheden. Aldaar bezocht hij van 1979 tot 1983 de Burgermeester de Bruin MAVO. In 1984 vervolgde hij zijn middelbare schooltijd in de 4e klas van de HAVO aan het Rhedens Lyceum te Rozendaal. Een HAVO diploma heeft hij nooit ontvangen, daar hij na de vierde klas van de HAVO naar de vijfde klas van het VWO is ‘weggepromoveerd’. In 1986 behaalde hij zijn VWO diploma, waarna hij zijn informatica studie aan de Katholieke Universiteit Nijmegen aanving. Die informatica studie rondde hij in mei 1990 af door het behalen van het doctoraalexamen. Vanaf juni 1990 tot juni 1993 is hij, als OIO in dienst van het NWO, aan de Katholieke Universiteit Nijmegen bij de Vakgroep Informatiesystemen gedetacheerd geweest in het kader van het EIS (Evolving Information Systems) project, om tenslotte van juni 1993 tot juni 1994 als AIO aangesteld te zijn geweest.

we apologise for the inconvenience