2008-08-22

A Generic Approach and Framework for Managing Complex Information

Essam Mansour

Technological University Dublin, essam.mansour@tudublin.ie

A Generic Approach and Framework for Managing Complex Information

By

Essam Mansour *B.Sc, M.Sc*

Supervisor: Dr Bing Wu

Thesis submitted to the Office of Postgraduate Studies and Research at the Dublin Institute of Technology in fulfilment of the requirements for the Degree of Doctor of Philosophy

June, 2008

School of Computing

Dublin Institute of Technology

Kevin Street, Dublin 8, Ireland.

Abstract

Several application domains, such as healthcare, incorporate domain knowledge into their day-to-day activities to standardise and enhance their performance. Such incorporation produces complex information, which contains two main clusters (*active* and *passive*) of information that have internal connections between them. The *active* cluster determines the recommended procedure that should be taken as a reaction to specific situations. The *passive* cluster determines the information that describes these situations and other descriptive information plus the execution history of the complex information. In the healthcare domain, a medical patient plan is an example for complex information produced during the disease management activity from specific clinical guidelines.

This thesis investigates the complex information management at an application domain level in order to support the day-to-day organization activities. In this thesis, a unified generic approach and framework, called SIM (**S**pecification, **I**nstantiation and **M**aintenance), have been developed for computerising the complex information management. The SIM approach aims at providing a conceptual model for the complex information at different abstraction levels (*generic* and *entity-specific*). In the SIM approach, the complex information at the *generic* level

is referred to as a skeletal plan from which several *entity-specific* plans are generated. The SIM framework provides comprehensive management aspects for managing the complex information. In the SIM framework, the complex information goes through three phases, specifying the skeletal plans, instantiating entity-specific plans, and then maintaining these entity-specific plans during their lifespan.

In this thesis, a language, called AIM (**A**dvanced **I**nformation **M**anagement), has been developed to support the main functionalities of the SIM approach and framework. AIM consists of three components: AIMSL, AIM ESPDoc model, and AIMQL. The AIMSL is the AIM specification component that supports the formalisation process of the complex information at a *generic* level (skeletal plans). The AIM ESPDoc model is a computer-interpretable model for the *entity-specific* plan. AIMQL is the AIM query component that provides support for manipulating and querying the complex information, and provides special manipulation operations and query capabilities, such as replay query support.

The applicability of the SIM approach and framework is demonstrated through developing a proof-of-concept system, called AIMS, using the available technologies, such as XML and DBMS. The thesis evaluates the the AIMS system using a clinical case study, which has applied to a medical test request application.

Declaration

I certify that this Thesis, which I submit for examination for the award of the degree of Doctor of Philosophy, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

This Thesis was prepared according to the regulations for post graduate study by research of the Dublin Institute of Technology, hereafter referred to as the Institute, and has not been submitted in whole or in part for an award in any other institute or university.

The Institute has permission to keep, to lend, or to copy this Thesis in whole or in part, on condition that any such use of the material of the Thesis is duly acknowledged.

.....

Essam Mansour

Dublin, Ireland

June 2008

Acknowledgements

"In the name of GOD (Allah), Most Gracious, Most Merciful. Praise be to GOD, the Cherisher and Sustainer of the worlds; Most Gracious, Most Merciful; Master of the Day of Judgement. Thee do we worship, and Thine aid we seek. Show us the straight way," [Quran: Al-Fatiha (The Opening)]. "and my success (in my task) can only come from GOD. In Him I trust and unto Him I turn (repentant)." [Quran: Hud,88]. I am not able to fulfil the due thanks to GOD, but I seek his forgiveness and that GOD assists me in thanking His Majesty.

"And your Lord has commanded that you worship none but Him, and that you be kind to parents. If either or both of them reach old age with you, say not to them (so much as) "Ugh" nor chide them, and speak to them a generous word. And make yourself submissively gentle to them with compassion, and say: O my Lord! have compassion on them, as they brought me up (when I was) little." [Quran: Al-Isra 23-24]. I am deeply indebted to my father for accepting that I pursue my PhD away from him. And May Allah bless my mother's soul and join all of us in the Paradise. I am grateful to my wife, ElKhansaa, and my children, for their encouragement and understanding. Thanks to my brothers and sisters, Duaa, Teama, Nesreen, Moataz and Mostafa.

I am deeply indebted to my supervisor, Dr Bing Wu, for many insightful conversations during the development of the ideas in this thesis, and for helpful comments on the text. Dr Wu has supported me not only by providing research guidelines and advices over almost four years, but also academically and emotionally through the rough road to finishing this thesis. And during the most difficult times when writing this thesis, he gave me the moral support. I also extend my sincere gratitude to Dr Kudakwashe Dube who has had a significantly positive influence on my development through his role as my research co-supervisor. His patience and willingness to discuss the minutiae of the different obstacles I encountered while working on this project were invaluable.

I want to thank the the Office of Postgraduate Studies and Research of the Dublin Institute of Technology (DIT) for sponsoring me to carry out this thesis and to publish my research findings. I am grateful to Dr Janet Carton, Ms Nicole O'Neill and Ms Denise Farrell and all the staff in the Office of Postgraduate Studies and Research for their kindly helping and supporting me over my PhD study period.

I am very grateful to Prof Brendan O'Shea, Mr Dave Carroll, Mr Paul Kelly, Ms Denise Murray and the staff of the School of Computing, who were very helpful to me during my study period. I am also grateful to Dr Fred Mtenzi for his encouragement, invaluable advices and interest in my progress throughout the study period. I also extend my sincere gratitude to the technicians of the School of Computing, specially Mr David Ng and Mr Michael Gleeson.

I would like to thank Dr Jamie Brett Stevens, a researcher in Astrophysics at the University of Tasmania, Australia, for providing me the style file of his PhD thesis.

I am deeply indebted to my MSc supervisor, Prof Mohamed El-Sharkawi, who taught me the craft of research. I also extend my sincere gratitude to all the staff

of the Faculty of Computers and Information, Cairo University, Egypt, where I did my undergraduate and master studies. I specially thank Professors, Reem Bahgat, Mokhtar Boshra, Ali Elbastawesy, Salwa Elgamal, Aly Fahmy, Ibrahim Farag, Ehab Hasaneen, Galal Hassan, Osman Hegazy, Sherif Mazen, and Khaled Mostafa.

Thanks for all my friends specially Ali Ahmed, Mohammed Al-Kalbani, Mohammed Al-Kateb, Amr Arisha, Ahmad Awad, Hosam Bayomy, Mohamed El Wakil, Mohammad El-Ramly, Waleed Elrawy, Gomathy Ramaswamy, Suhaib Fahmy, Mostahfizz Gani, Mohamed Ghaleb, Xiang Han, Mohamed Maher, Sherif Sakr, Yasser Salem, and Erqiang Zhou.

I would like to express my gratitude to all those who gave me the possibility to complete this thesis.

. . . . . .

Essam Mansour

Dublin, Ireland

June 2008

Glossary and Abbreviations

active part	is one of the main clusters or parts of the complex information and determines the recommended procedures or actions that should be taken as a reaction to specific situations, 1
passive part	is one of the main clusters or parts of the complex information and determines the information that describes the situations invoking the *active* part and other descriptive information plus the execution history of the complex information, 1
active XML	XML-based ECA rule languages, 24
ADBMS	Active Database Management System, 40
AIM	is a complex information specification and query language and is an acronym for **A**dvanced **I**nformation **M**anagement, 86
AIMQL	is the AIM query and manipulation sub-language, 119
AIMQL Replay Language	is a language that plays over again the evolution (history) of the complex information, 132

AIMS	is a proof-of-concept system for managing the Complex Information, and utilizes the available XML technologies and database systems as a base for its functionality, 141
AIMSL	is the AIM specification sub-language, 87
AIMSL ECA Rule Paradigm	provides a temporal support for the ECA rule paradigm at an application domain level, 98
Business Process Management (BPM)	encompasses methods, techniques and tools to design, enact, control, and analyse business processes involving organizations activities, 21
Business Process Modelling	focuses on process formalisation, validation and verification to model domain knowledge as processes with maintained control flow and order, 21
Clinical Guidelines	provide guides for clinicians and patients in determining recommended strategies for managing and monitoring the patients condition, 17
CoAX	is a **Co**mparative fr**A**mework for **X**ML-Based ECA Rule Languages, 24
Complex information (CI)	(*From Existence Perspective*) is interconnected and clustered information produced during day-to-day organisation activities, which incorporate domain knowledge to standardise and enhance their performance, 1
Complex information (CI)	(*From Nature Perspective*) contains two main clusters (*active* and *passive*) of information that have internal connections between them, 1

Complex information Management	in this thesis, focuses on specifying the complex information at different abstraction levels (generic and entity-specific), instantiating entity-specific instances and maintaining these instances by providing execution, manipulation and query support with emphasising the demand to history and replay facilities, 3
DBMS	Database Management System, 119
ECA	Event-Condition-Action, 12
ECA rule paradigm	is paradigm with a reactive semantics; when an event occurs, check the condition and execute the action only if the condition is evaluated to true, 15
entity-specific (ES) plan	is a conceptual model for the complex information at an entity-specific level, 66
Event-driven Process Chain (EPC)	is an ordered graph of events and functions, 21
Extensible Markup Language (XML)	is a general-purpose specification for creating custom markup languages, 13
patient plan	is an example of the complex information, which is produced during the disease management from a specific clinical guideline to suit a particular patient, 2
Protocol	in this thesis is a logical model for the skeletal plan, 92
RDB	relational database, 148

SIM	is an approach and framework for managing complex information and stands for **S**pecification, **I**nstantiation, and **M**aintenance, 63
SIM approach	aims at providing a conceptual model for the complex information at an application domain level with different abstraction levels, 66
SIM framework	is a management framework for the complex information and consists of three planes, specification, instantiation and maintenance, 72
skeletal plan	is a conceptual framework or model for the complex information at a generic level, 66
Temporal Active XML	is a combination of temporal database, ECA rule paradigm and XML, 4
The AIM ESPDoc	provides a computer-interpretable or logical model for the entity-specific (ES) plan, 112
TRME	is an intermediate model that translates AIMSL rules into a pure SQL triggers, and is an acronym for **T**emporal **R**ules *M*ade **E**asy, 153
TXME	is a temporal XML data model that implements the AIM ESPDoc model, and is an acronym for **T**emporal **X**ML **M**ade **E**asy, 168
XQuery	is an XML query language that provides means to extract and manipulate data from XML documents, 15

Contents

List of Figures

List of Tables

1

Introduction

This chapter is organised as follows: Section 1.1 discusses the existence of the complex information and its nature; Section 1.2 presents the main focus of the thesis in terms of research problem, implementation method and challenges; Section 1.3 introduces the background of this research; The aim, objectives and scope of this thesis are discussed in Section 1.4; The expected benefits of the thesis and its organisation are presented in Sections 1.5 and 1.6 respectively.

1.1 Complex Information: Existence and Nature

In the context of this thesis, complex information (CI) is referred to as interconnected and clustered information produced during day-to-day organisation activities, which incorporate domain knowledge to standardise and enhance their performance. For each entity, to which these activities are applied, the complex information contains two main clusters of information that have internal connections

between them. These clusters are: 1) an *active* cluster of information that determines the recommended procedure, which should be taken as a reaction to specific situations; and 2) a passive cluster of information that describes these situations plus the execution history of the recommended procedure.

Complex information exists in several domains. In healthcare domain, the Clinical Guidelines (Field and Lohr 1992) are instantiated to a specific patient in the activity of disease management (Shahar 2002). In agriculture, the Good Agricultural Practices are instantiated to a specific animal in the activity of animal production management (FAO 2003). In stock exchange, the Best Execution Guidelines are instantiated for a specific customer in the activity of customer securities order management (EAMA 2002).

In these activities, an instantiation process produces a plan for managing a particular entity in a specific activity. According to its specific domain, that entity could be a patient, animal, or customer order. Such a plan is an example of complex information that consists of the following main components:

- the domain information or data items that is relevant and is therefore required to be monitored in the activity;
- recommended procedures that are inherited from domain knowledge, and are applied in consideration to the users preferences or situations;
- A descriptive information about the plan or reference material associated with the specific area of focus; and
- the history of the plan evolution and experience arising from daily practice of using best practices for this particular entity.

In the healthcare domain, the patient plan is an example of the complex information, which consists of:

- particular healthcare information that is monitored in the patient record;
- Clinical Guidelines that provide suggestions and provide guidance to patients and clinicians in making decisions about disease management in consideration of the variations in the monitored patient healthcare information;
- descriptive and didactic information for the Clinical Guidelines and procedures as applied to the patient; and
- the care plan progression history that is required in enhancing and reviewing the applied information and knowledge from the Clinical Guideline.

1.2 The Complex Information Management

The complex information management at an application domain level is the main topic to be investigated in this thesis. In this thesis, the complex information management focuses on specifying the complex information at different abstraction levels (generic and entity-specific), instantiating entity-specific instances and maintaining these instances by providing execution, manipulation and query support with emphasising the demand to history and replay facilities. The thesis research questions are:

(1) what is a suitable and practical *way* to model and manage the complex information as it is seen by the domain users?

(2) according to this *way* how to facilitate the complex information management using a high level and declarative *language*?

(3) how to utilize the available technologies, such as XML and database systems, to demonstrate that the adopted *way* supported by this *language* can be applied in practice?

1.2.1 Research Problem

The problem of this research is three-fold. The first problem is the need to a generic approach for modelling the complex information at an application domain level. Applying this approach to activities, such as disease management, animal production management and/or securities order management, provides a computerised patient plan, animal production plan and/or customer order execution plan, respectively. The computerised version of these plans and their components should be managed as a first class object.

The second problem is the need to a management framework for computerising the complex information. The framework is to specify the complex information at different abstraction levels in order to support a variety of domain entities. Consequentially, the complex information is to be defined initially for a general group of entities, then instantiated to a particular entity. For example, defining a generic plan for a group of diabetes patients and instantiating this plan for particular patients support the varieties between the diabetes patients. The framework should provide the functionality supporting the maintenance of the complex information, such as maintaining the patient plan during its lifespan. The maintenance support emphasises the need to record the complex information execution history and replay this history. Recording and replaying the execution history provide a motion picture that depicts the evolution of the complex information. Such motion picture facilitates the review and decision-support capabilities in the organization.

The third problem is the need to an implementation method realizing the approach and framework as a unified and high-level method using the available technologies. The adopted technologies are to be seamlessly integrated and easily incorporated with the domain application systems in order to demonstrate the complex

information management in practice.

1.2.2 An Implementation Method

This thesis adopts the combined application of XML, the ECA rule paradigm, and a temporal database mechanism supported by database systems, as an implementation method for the approach and framework developed by the thesis. This combination of temporal database, ECA rule paradigm and XML presents the concept of Temporal Active XML. The hypothesis of the thesis is that the Temporal Active XML method supported within database systems is an effective and practical tool for facilitating and realizing the management of the complex information. This hypothesis is supported by the following:

- the active database, which a database includes triggering mechanism, is considered as a connection between systems effectively handling data storage and information retrieval, and systems with the power of a rule language in monitoring changes and expressing complex inference mechanism (Caironi et al. 1997). Database systems are widely used as a base for managing information domains. That means an easy integration between systems managing the complex information and systems managing domain information;
- the ECA rule paradigm has been proven to be effective in supporting the specification of best practices (Clayton et al. 1989; Caironi et al. 1997; Wu and Dube 2001);
- the ECA rule paradigm and XML are seamlessly integrated and easily incorporated in research proposals, such as (Bonifati 2000; Kiyomitsu et al. 2001; Schrefl and Bernauer 2001; Abiteboul et al. 2002), and in modern database systems, such as DB2 (Nicola and Linden 2005) and Oracle (Mark Scardina 2004; Zhen Hua Liu 2005);

- regarding the distributed management, XML provides capabilities, such as heterogeneity, extensibility, and flexibility, that support the distributed management (Mller and Schwartzbach 2006); and
- the temporal database provides support for keeping the history and tracking the evolution of domain information (Tansel et al. 1993).

1.2.3 Research Challenges

Complex information management poses major challenges for information management. These challenges could be classified into *intellectual* and *practical* categories. The *intellectual* challenges are: 1) the challenge of modelling conceptually the complex information at different abstraction levels (generic and entity-specific); 2) the challenge of supporting the instantiation process, on which the complex information is defined to suit a particular entity, and 3) the challenge of maintaining the complex information evolution history and replaying it to provide a motion picture of the complex information.

The *practical* challenges are based on the adopted implementation method, which realizes the *active* part of the complex information using XML-based ECA rules and the passive part as temporal XML. The first challenge is to provide an execution mechanism for the complex information. That needs to translate the complex information from a generic level into an entity-specific level. That translation maps the platform-independent rules of the generic version into platform-dependent rules in the entity-specific version. The major challenge here is that database systems do not incorporate a comprehensive implementation of the ECA rule paradigm, which is adopted by the thesis as implementation method. Instead, the database systems provide a basic triggering mechanism, which has a number of limitations in its support of the ECA rule components (Ceri et al. 2000). For example in the complex

information, the events are not limited to the basic events of the triggering mechanism that are based on the occurrence of the INSERT, DELETE and UPDATE operations. The main challenges here are to provide an extension to

- the event component to provide support for the time-based and domain-specific events;
- the condition component to support the specification and evaluation of temporal conditions in the ECA rules; and
- the action component to allow detached actions to be performed externally and at time point after the rule has been executed.

The second challenge is to keep the execution history in order to review and analysis the evolution of the complex information. The core challenge here is to provide temporal extensions to the XML data model. The difficulty here is to provide a temporal XML data model that is compatible with the XML data model, in order to re-use the XML support provided by the database systems. The third challenge is to facilitate the manipulation and query of the complex information as a first class object. That means the complex information is subject to the same manipulation and query operations, as domain information, plus special operations that handle the rules and reviewing the execution history.

1.3 Background

The most related research areas to this thesis research are the workflow and computerised clinical guidelines. Both areas focus on specifying and executing the *active* part of the complex information. That part determines the recommended procedure or action as a reaction to specific situations. The focus of the workflow approaches is to model and manage only the *active* part as business processes. The approaches

of computerised clinical guidelines overlook the need to specify and manage the patient plan (complex information), which is produced by applying a specific clinical guideline to a particular patient.

This thesis work differs from all these approaches by providing a generic approach and framework for managing the complex information at a platform-independent and application domain level. The thesis provides a unified management environment that provides support to specify and formalize the complex information at a generic level, instantiate complex information instances, such as patient plan, execute these instances, keep the execution history incorporated into each instance, manipulate and query all these pieces of information at a high and declarative level.

This thesis reports the second stage of on-going research of KCAMP Group led by Dr Bing Wu at Dublin Institute of Technology. The first stage of this research work has developed a framework with a declarative language PLAN (Wu 1998; Wu and Dube 2001) for specifying clinical guidelines of reactive applications, such as clinical test request application. Furthermore, a prototype system TOPS (Dube 2004) was developed using relational active database to implement the framework and language.

The PLAN specification is represented in plain text. Querying and manipulating a text file is limited to specific functions, such as find and replace functions, respectively, that are provided within text editors. It is very important to provide query and manipulation support for the domain knowledge specification. In order to provide such support, TOPS (Dube 2004) maps the PLAN specification plain text into database schema to be stored and managed using the DBMS. However, mapping the PLAN specification into relational database schema decomposes the specification into several tables. Therefore, it is not easy to deal with the specification as one document, as it is in the real life. Moreover, it is not easy to exchange

the specification between heterogeneous systems. TOPS did not provide multi-level of abstraction nor a model for the complex information.

1.4 Research Aim, Objectives and Scope

This thesis aims at providing an applied approach for facilitating the management of the complex information at an application domain level. The main objectives of this study are to:

- develop a generic approach for modelling the complex information and unified framework for managing the complex information;
- develop a high level declarative language for facilitating the management of the complex information;
- develop a proof-of-concept system using the available technologies to demonstrate the applicability of our work; and
- evaluate the system using a clinical case study.

The approach and framework of the thesis are restricted to applications that naturally take the form of reactive applications that monitor events of interest to domain users, and respond to changes in situations by issuing alerts, reminders, requests, and/or observations to the domain user. Our approach and framework do not provide recommendations on courses of action, but rather provide the necessary information needed to make informed decisions.

1.5 Expected Research Benefits

This thesis contributes in a number of ways to the research of the information management. The major contributions of this thesis are:

- a generic approach and framework, called SIM. The SIM approach provides a conceptual model for the complex information at different abstraction levels; generic and entity-specific. The SIM framework classifies the requirements of the complex information management into three generic planes; specification, instantiation and maintenance.

- an advanced language, called AIM, for supporting the SIM approach and framework. This language is based on XML and the ECA rule paradigm and consists of three main components; specification component (AIMSL), instantiation model (AIM ESPDoc) and query component (AIMQL).

- a proof-of-concept system, AIMS, for demonstrating that the available XML and database systems could be extended to support the SIM approach and framework and implement the AIM language.

The minor contributions of this thesis are:

- the *TRME* model, which extends the DBMS triggering mechanism to support the advanced features of AIMSL, such as time-based ECA rules. Using the *TRME* model, the AIMSL rules are translated into pure SQL triggers managed by the DBMS.

- the *TXME* model, which extends the XML support provided by the modern DBMSs to implement the AIM ESPDoc model. The *TXME* model is consistent and compatible with both XML Schema and the XML data model. Using the *TXME* model, the complex information could be stored and retrieved using the modern DBMSs.

- an evaluation of the AIMS system using a clinical case study, which focuses on evaluating the AIMS execution mechanism based on the *TRME* model, the

AIMS repository based on the *TXME* model, and the AIMS queries performance.

1.6 Thesis Organization

This thesis is organized as follows. Chapter 2 reviews the most related approaches proposed in the area of workflow management and the computerised clinical guidelines. The implementation method adopted in this thesis is based on a combination of XML and ECA rule paradigm. Chapter 2 develops a framework for comparing and analysing the available XML-based ECA rule languages.

Chapter 3 presents a generic approach and framework, called SIM, for modelling and managing the complex information. Chapter 4 presents a high-level declarative language, called AIM, for facilitating the management functions provided by SIM. In Chapter 5, a proof-of-concept system, called AIMS, is presented. Chapter 6 presents a case study in which AIMS is used to manage patient plans existing in a clinical test request application. Chapter 6 is concluded by discussing the evaluation results of the case study. The thesis summary and future work are discussed in Chapter 7.

2

Related Work and Analysis of XML-Based ECA Rule Languages

This chapter reviews relevant approaches addressing the complex information management, and presents a framework, called CoAX, developed to compare XML-based ECA rule languages. The thesis adopts the clinical guidelines management used for test request protocol as an application domain, in which the complex information management is demanded. Therefore, the chapter discusses the approaches for computerising the clinical guidelines management. This chapter provides a classification for the clinical guidelines approaches that are based on the ECA rule paradigm and XML. The chapter also presents a brief literature review for the workflow approaches, which are used to support the active part of the complex information as a step for incorporating domain knowledge into organization activities.

The Event-Condition-Action (ECA) rule paradigm incorporated into XML is adopted as a main method for realizing the thesis approach and framework for

managing the complex information produced from incorporating domain knowledge into organization activities. The chapter provides a brief introduction about the XML language and the ECA rule paradigm.

The chapter presents a framework, called CoAX, for comparing the XML-based ECA rule languages. The CoAX framework outlines the main features of the XML-based ECA rule languages, analyses and compares six typical XML-based ECA rule languages, which have been developed by several institutions. Moreover, CoAX offers a classification of these languages, depicting their evaluation. These languages range from ones, which standardize the ECA rules representing a specific domain knowledge and more properly targeted to relational database, to languages, which extend the W3C consortium for a standard XML query language.

The rest of this chapter is organized as follows: Section 2.1 is an introduction for the XML language and the ECA rule paradigm; Section 2.2 classifies the computerised clinical guidelines approaches based on the ECA rule paradigm and XML, and discusses the support provided to the clinical complex information; Section 2.3 presents the workflow approaches, and discusses the differences between the research addressed in this thesis and the workflow research; Section 2.4 presents the CoAX framework; Section 4.4 summarizes the chapter.

2.1 Background: the XML Language and ECA Rule Paradigm

This section introduces the main technologies used to support the development of the research presented in this thesis. These technologies are mainly the XML language and the Event Condition Action (ECA) rule paradigm.

2.1.1 The XML Language

The Extensible Markup Language (XML) is a general-purpose specification for creating custom markup languages. The XML language was developed by Bray et al. (1998). The most recent recommendation of the XML language has been presented by Bray et al. (2008). XML become the prime standard for data exchange on the Web (Arciniegas 2000). XML is a language to represent semi-structured data, which refers to data with some of the following characteristics:

- The schema is not given in advance and may be implicit in the data;
- The schema is relatively large;
- The schema is descriptive rather than prescriptive, i.e., it describes the current state of data, but violations of the schema are still tolerated;
- The data is not strongly typed, i.e., for different objects; the values of the same attribute may be of different types.

Any XML document should be a Well-formed document (Arciniegas 2000), which is a document conforming to all of XML's syntax rules. The XML document might additionally conform to some semantics rules. These rules are either user-defined, or included as an XML schema (Fallside and Priscilla 2004).

A valid XML document means that the document has been validated against a rule set, such as a Document Type Definition (DTD) or an XML Schema (Fallside and Priscilla 2004). An XML document is not considered valid unless it has a DTD or XML Schema, and the document meets the constraints in that schema. DTDs are a type of schema for describing the data structure of an XML document. DTD could be used to specify the types of the child element, the order and number of times the element may occur within a document, and the default value. DTD is

one of the technologies of SGML, so DTDs are not designed specifically for XML and therefore although some of the syntax and structures of DTD might seem very convoluted, they are the result of adapting the SGML technology for XML. An XML Schema defines a class of XML documents by providing constraints on both structure and content. XML schemas offer an alternative to describing an XML grammar using DTDs. The main advantage of XML Schema is that schemas are actually XML documents.

XQuery provides means to extract and manipulate data from XML documents or any data source that can be viewed as XML, such as relational databases or office documents (Walmsley 2007). Chamberlin et al. (2001) proposed the first working draft of the XQuery language. The most recent XQuery recommendation is presented by Boag et al. (2007). XQuery is supported with some programming language features (Walmsley 2007). The XQuery language is a SQL-like language with the main "FLWOR expression" for performing joins. A FLWOR expression is constructed from the five clauses after which it is named: FOR, LET, WHERE, ORDER BY, RETURN. The reader is refered to Walmsley (2007) for more details regarding XQuery.

2.1.2 The ECA Rule Paradigm

The Event Condition Action (ECA) rule paradigm (Widom and Ceri 1996; Paton 1999) refers to the structure of active rules in event driven architecture and database systems. The general structure of the ECA rule paradigm is:

- the event part specifies the signal that triggers the invocation of the rule,
- the condition part is a logical test that, if satisfied or evaluates to true, causes the action to be carried out,
- the action part consists of updates or invocations on the local data.

Most modern database systems support the ECA rule paradigm through a triggering mechanism. In the relational database systems, this triggering mechanism implements the SQL triggering language (Kulkarni et al. 1999). Some recent database systems (Mark Scardina 2004), which provide XML storage and retrieval support, extended the SQL language (Kulkarni et al. 1999) with XML functions, which is known as SQL/XML (Andrew and Melton 2002; Sql/Xml 2003) language. The SQL/XML language in most database systems is incorporated with their triggering mechansim, such as in Oracle (Mark Scardina 2004; Zhen Hua Liu 2005) and DB2 (Nicola and Linden 2005; Chen et al. 2006).

2.2 Computer-Based Clinical Guidelines and Patient Plan Management

This section provides a brief review for the computerised clinical guidelines. The focus of this review is on the use of the XML and event-deriven approach to support the computerised clinical guidelines. Clercq et al. (2004) and Dube (2004) have analysed and evaluated several approaches for computerising the clinical guidelines management. The reader is referred to these references for more details about the computerised clinical guidelines approaches. Most of the computerised clinical guidelines approaches focus on specifying and executing clinical guidelines, and give little attention to the query and manipulation support (Clercq et al. 2004).

Figure 2.1 illustrates the author's view on the main dimensions of the XML and event-driven support for the computerised clinical guidelines. These dimensions are *clinical guidelines*, *clinical events*, *XML*, and *ECA rule paradigm*. Clinical guidelines should ideally be executed as soon new or extra patient information, which generally represents some changes in the patient circumstances, becomes available. It would

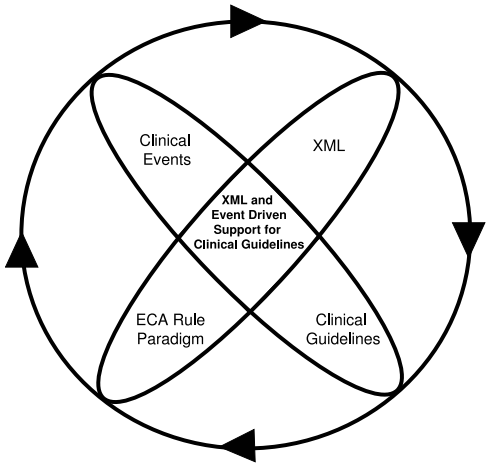

Figure 2.1: The dimensions of the XML and event-driven support for the computerised clinical guidelines.

benefit the patient, if clinicians could be informed of the recommendations from clinical guidelines based on the contents of the medical record. Thus, the event-driven approach appears to be one of the suitable ways to support effective and efficacious computerisation of clinical guidelines.

2.2.1 Clinical Guidelines and Events

The clinical guidelines provide guides for clinicians and patients in determining recommended strategies for managing and monitoring the patients condition (Field and Lohr 1992). A computerised clinical guidelines management is one of the suggested methods for improving and enhancing the health care services (Grimshaw and Russell 1993; Dart et al. 2001; Shahar 2002; Votruba et al. 2004).

The monitoring and detection of clinical events play key roles in the practice of disease management and patient care (Hripczak et al. 1996). In a pioneering study that used a computer to detect and respond to clinical events, (McDonald 1976) concluded that computer detection and response to simple clinical events would have a positive effect on the behaviour of clinicians and build a foundation for more complex clinical event detection. Studies of clinical events occurring before and

during disease progression help to inform treatment and deepen understanding of disease progression (Khanda et al. 2000). Hence, clinical events could be seen as a core driver to clinical practice guidelines and protocols.

2.2.2 The XML and ECA Rule Paradigm Support for Clinical Guidelines

Several research efforts propose XML-based languages for formalising clinical guidelines (Jones et al. 2005; Georg and Jaulent 2007; Wainer et al. 2008; Casteleiro and Diz 2008). As in other approaches, these research works focus on specification and execution while providing a little or no support for manipulation or querying of clinical guideline information. Most efforts in supporting information sharing in guideline management approaches have been concentrating on making the formal clinical guidelines specification sharable across healthcare institutions (Greenes et al. 2001; Ciccarese et al. 2003). The problem of sharing clinical guidelines specifications has been dealt with in literature (Pattison-Gordon et al. 1996; Greenes et al. 2001; Dart et al. 2001). However, these works did not consider the specification and the execution of clinical guidelines within a computer-supported collaborative environment.

Furthermore, the means for sharing knowledge and information in patient care practice continues to be based mainly on paper-based methods. Thus, patient information continues to be shared between collaborating clinicians primarily through referral and clinical notes. However, significant research efforts have been expended into supporting the sharing of electronic patient records among healthcare institutions by Grimson et al. (1998) and Halamka et al. (1998). Little effort has so far been expended in supporting information manipulation, sharing and collaboration with respect to the key aspects of guideline management.

In order to support distributed clinical event monitoring, the clinical event is specified using XML. This combination is used to implement a wide range of clinical event notification system that provides notification via remote procedure calls, such as work done by Arabshian and Schulzrinne (2003)

The event-driven approach based on ECA rule paradigm has been adopted in computerised clinical guidelines systems. TOPS (Dube 2004), which was developed by the author's research group, the Arden Syntax (Clayton et al. 1989) and HyperCare (Caironi et al. 1997) are the major relevant works that computerised clinical guidelines by following the event-driven approach. Arden Syntax does not distinguish between the generic specifications of clinical guidelines and the generated instance. HyperCare computerises a clinical guideline without providing a generic mechanism to be applied to other guidelines. Like other computerised clinical guideline approaches, both the Arden Syntax and HyperCare provide a little or no support for manipulation and querying clinical guideline information. Both Arden Syntax and HyperCare do not provide support for creating a medical plan. They manage the clinical guidelines at the rule level.

2.2.3 Discussion: Patient Plan Management

A patient plan represents an instance of a specific clinical guidelines applied to a particular patient. In healthcare domain, the patient plan could be seen as an example of the complex information, which contains *active* and *passive* parts. The *active* part determines the recommended procedure that should be taken in specific situations. The *passive* part determines the information that describes these situations and other descriptive information.

Computerizing clinical guidelines mainly covers the specification and execution the guidelines. Consequentially, the computerised clinical guidelines approaches

focus on the *active* part of the complex information. These approaches overlook the need to specify and manage the patient plan as one distinct entity; as it is seen in the healthcare domain.

This thesis provides an approach and framework for not only specifying and executing the complex information, but also manipulating, querying and distributing the complex information. Applying this approach and framework to the patient plan facilitates the patient plan management at an application domain and end-user level. Therefore, the patient plans are to be managed under a unified framework that provides support to specify the clinical guidelines, instantiate patient plans using these guidelines, execute these plans, keep the execution history incorporated into each plan, manipulate and query all these pieces of information at a high and declarative level.

The approach presented in this thesis facilitate the dissemination of not only the clinical guidelines specification but also the patient plans. In this approach, the patient plan represents an application case of a specific clinical guideline. The ability to review the execution history of the patient plan within any time period is supported in the Author work. Reviewing the execution history is leading to a new research trend, that is to investigate into mining the patient plans to enhance and discover new clinical guidelines.

2.3 Workflow and Business Process Approaches

In the area of workflow management, several research efforts have addressed the problem of incorporating domain knowledge into organization activities. In this thesis, the workflow approaches are classified into three categories, Business Process Management (BPM), Adaptive Workflow and Process Mining. The process mining category is ignored because it is not strongly related to this thesis. The first two

categories and the distinguishing features of this thesis research focus are discussed in the following sub-sections.

2.3.1 Business Process Management

Business Process Management (BPM) encompasses methods, techniques and tools to design, enact, control, and analyse business processes involving organizations activities (van der Aalst et al. 2003). A survey highlighting the life-cycle of the BPM has been presented by van der Aalst et al. (2003). In BPM, the focus is on managing domain knowledge as business processes, which are modelled using different modelling approaches.

The business process modelling focuses on process formalization, validation and verification to model domain knowledge as processes with maintained control flow and order (Lu and Sadiq 2007). The business process modelling languages are classified into graph-based languages, such as van der Aalst and ter Hofstede (2005), or rule-based languages, such as Knolmayer et al. (2000a). Lu and Sadiq (2007) have presented a comparative analysis of the business process modelling languages.

Most of the graph-based languages are based on Petri Nets (Lu and Sadiq 2007). Two surveys on the Petri Nets-based languages and their applications in workflow modeling have been presented in Janssens et al. (2000); Kiepuszewski et al. (2003). Other graph-based languages, such as SAP reference model (Curran et al. 1997; Keller and Teufel 1998), are based on Event-driven Process Chain (EPC), which is an ordered graph of events and functions (Verbeek and van der Aalst 2006). The business process modelling, such as van Dongen et al. (2007); Rosemann and van der Aalst (2007), provide formalization and validation models for business processes based on the SAP reference model.

Most of the rule-based languages are based on business rules (Knolmayer et al.

2000b). The business rules are supported by using the ECA rule paradigm, such as in Müller et al. (2004), Inference Rules, such as in Zeng et al. (2002), and Web Services, such as in Orriëns et al. (2003). The support of the ECA rule paradigm to the business processes is discussed by Bry et al. (2006).

The main advantages of the ECA rule paradigm are the flexibility, dynamism and adaptability to the dynamic changes in the business processes (Bry et al. 2006; Lu and Sadiq 2007). In workflow systems, the ECA rules are implemented using several technologies such as active object oriented database systems as in Kappel et al. (1998, 2000) and active database systems as in Müller et al. (2004).

The use of the ECA rule paradigm for implementing the workflow systems facilitates the integration with the systems managing the domain information, because most of these systems use the DBMSs that support the ECA rule paradigm. The ECA rule paradigm is utilized to implement workflow management system, such as Wang et al. (2006) that supports the advanced resource reservation.

2.3.2 Adaptive Workflow

One of the workflow research trends is to support adaptive workflow, which is able to change when necessary in order to improve the exception handling and deal with failure situations that may occur during workflow execution (Müller 2002). The workflow adaptation is supported using several approaches, such as data-driven process modelling (Müller et al. 2006), case-based reasoning (Weber et al. 2006), templates-oriented (Gottschalk et al. 2007), variability model (Rosa et al. 2007) and agent-based models (Müller et al. 2004). The common feature among these approaches is that these approaches cope with the logical failures occurring during workflow execution.

2.3.3 Discussion

Implementing domain knowledge into workflow systems forces organizations, which generally do not have formal procedures, to conform to a single standard. Deviation from this standard requires a change to the systems of these organizations (Rinderle and Reichert 2007). The workflow approaches focus only on the active part of the complex information. That means the active part and the passive part of the complex information are not detached.

Most of workflow approaches provide modelling languages that model the active part of the complex information as business processes. It is easier for the end user to deal with the complex information as one distinct entity; as it exists in the real-world. For example, clinicians deal with medical patient plan as one distinct entity that includes information about reacting to specific situations and information about the patient and the execution history of the plan. In workflow, the focus only on the information about reacting to specific situations.

The adaptive workflow approaches deals with exception handling and logical failures during workflow execution. Instead, the complex information adaptation is to adapt the general medical plan incorporated from the domain knowledge to a specific patient before execution. These workflow approaches provide little or no support for manipulating the active part of the complex information using a manipulation language.

The author's approach differs from workflow approaches in that workflow approaches address the specification aspects with little or no support to query and manipulate all aspects of the information in a unified manner. However, the author's approach aims at computerising the management of the complex information using a unified framework that provides support to specify, execute, query, and

manipulate the complex information.

The major challenge addressed in this thesis is to provide an approach and framework that give flexibility in incorporating domain knowledge into the day-to-day organization's activities, and managing the instantiated plans, such as patient plans, at the domain entity level. This thesis distinguishes from all workflow approaches by providing a generic approach and framework for managing the complex information at a platform-independent, domain, and high level under a unified management environment. Consequentially, the workflow could be used to support the low-level implementation of our approach and framework.

2.4 CoAX: A Framework for Comparing XML-Based ECA Rule Languages

Novel languages that incorporate the ECA rule paradigm into XML have been proposed, such as in Bonifati et al. (2002), Bailey et al. (2002b) and Abiteboul et al. (2002). In this thesis, XML-based ECA rule languages refer to these novel languages. The ECA rule paradigm could be used to represent the event-based behaviour of the application domain (Caironi et al. 1997; Jenders et al. 1998; Dube et al. 2002; Mansour et al. 2007). The event-based behaviour performs actions or reactions in response to events (Widom and Ceri 1996; Paton 1999). The semantics of the ECA rule paradigm is that when an event E occurs, evaluate a condition C, and if the condition is satisfied, then execute an action A (McCarthy and Umeshwar 1989; Widom and Ceri 1996; Paton 1999).

The XML-based ECA rule languages aim at reacting according to events that happen to XML data or using XML to standardize and unify the specification of ECA rules. The XML-based ECA rule languages, as languages for supporting the

active part of the complex information, have different shortcomings.

These languages:

- specify and execute rule individually. However, it is needed to specify, execute and manipulate the specification of the *active* part as one unit of cohesive and correlative rules that achieve a certain objective.
- provide a basic trigger implementation of the ECA rule paradigm. In order to support a comprehensive ECA rule paradigm, several features are needed, such as providing support for composite and temporal events, supporting the specification and evaluation of temporal conditions, and allowing not only the primitive actions, update, delete and insert, but also advanced actions, such as application defined and time-based actions.

ECA rule paradigm could be implemented using the SQL triggers. When shifting from relational databases to XML data, it is necessary to review the features of SQL triggers. In databases, SQL trigger is associated with a table and according to the update operations the trigger is activated. However, the XML-based ECA rule languages are of a tree structure within it the event-based rules among heterogeneous and distributed applications need to be supported.

This section presents a framework, called CoAX for analysing and comparing the XML-based ECA rule Languages. CoAX provides a comparative outline of the XML-based ECA rule features by analysing and comparing XML-based ECA rule languages that have been proposed. Moreover, CoAX offers a classification of these languages, depicting their evolution from several perspectives. The implementation approaches and technologies for the XML-based ECA rule languages are discussed. These languages range from ones that use XML format and are targeted to deal with relational database to the latest proposal of W3C consortium for a standard XML query language.

2.4.1 An Application Example

This sub-section introduces a case study that is used throughout Section 2.4 to exemplify the analysed XML-based ECA rule languages. This case study is originally proposed by Chandra and Arie (1994) and also used by Paton (1999). The Author has modified this case study to become suitable for the XML-based ECA rule languages. This case study is selected because the real world application greatly helps to understand the capabilities of the languages and determine the differences between these languages.

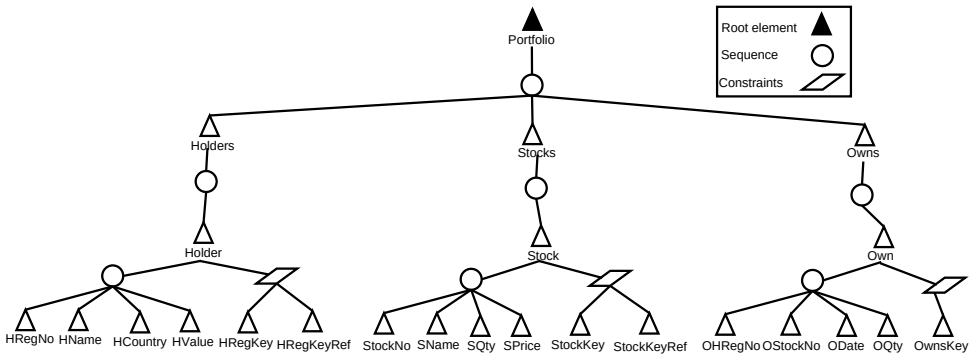

Figure 2.2: A tree digram for a portfolio XML schema.

Figure 2.2 illustrates the tree diagram of the XML Schema of a financial market portfolio. It consists of three element *Holder, Stock* and *Owns*. A *Holder* is an individual or organization that owns stocks. Every *Holder* has a unique registration number, name, country, and total value of stock held. An organization that has been floated on the stock market is represented by the element Stock, which has elements that record the organization's name, share price, the total number of shares available, and the unique identification number by which it can be referenced. The element *Owns* indicates that a *Holder* possesses *OQty* items of a particular kind of *Stock*. This application scenario has the following active semantics:

- Integrity Constraints:

Key Constraint. The elements *Holder, Stock* and *Owns* have key constraint *HRegKey, StockKey* and *OwnsKey* respectively. *HRegKey* is a key constraint that define *HRegNo* as a key for the element *Holder*, and *StockKey* is a key constraint that define *StockNo* as key for the element *Stock*. For the element *Owns*, *OwnsKey* defines a composite key that consists of *OHRegNo, OStockNo* and *ODate*.

Referential Integrity. The element *Holder* has a key reference, in which the *OHRegNo* element refers to the key *HRegNo*. The element *Stock* has a key reference, in which the *OStock* element refers to the key Stock.

- Web Content Rules:

  Rule 1. Delete Cascade Rule. When a holder is deleted, delete its *owns* from the element *Owns*.

  Rule 2. The organizations that has been floated on the stock provide their share price on their web site. The value of *SPrice* element should be equal to the last up to date price on the organizations' web site.

  Rule 3. the web site of Portfolio XML database should add the content of read me XML document to the main page of the web site for a user who accesses at the first time or has accessed since more than 6 months ago.

- Business Rules:

  Rule 4. Do not allow the delete operation for a holder to proceed if that holder has a value > 0, and inform the system manager through email or SMS.

  Rule 5. Report the system manager by the holders who increase the possessed items of a particular kind of stock within last 6 months.

2.4.2 Brief Overview of the XML-Based ECA Rule Languages

This sub-section provides a brief overview of current available XML-based ECA Rule languages. The chosen languages are Active XQuery (Bonifati et al. 2002), ECA language for XML (Bailey et al. 2002b), Active XML or AXML (Abiteboul et al. 2002), Activeweb (Kiyomitsu et al. 2001), Active XML Schemas (Schrefl and Bernauer 2001) and ARML (Cho et al. 2002).

2.4.2.1 Active XQuery

Bonifati et al. (2002) have designed the Active XQuery language for extending the W3C proposed standard XQuery language. Active XQuery adapts the active database features in SQL3 to hierarchical nature of XML data. It is a user-friendly language in the XQuery style. A proposed implementation to it over XML views that created from relational database has been developed by Shao et al. (2004).

CREATE TRIGGER Trigger-Name
[**WITH PRIORITY** Signed-Integer-Number]
(**BEFORE**|**AFTER**)
(**INSERT**|**DELETE**|**REPLACE**|**RENAME**)+
OF XPathExpression (,XPathExpression)*
[**FOR EACH** (NODE|STATEMENT)]
[XQuery-Let-Clause]
[**WHEN** XQuery-Where-Clause]
DO (XQuery-UpdateOp|ExternalOp)

Figure 2.3: The syntax of an XQuery trigger.

Active XQuery Syntax. Figure 2.3 illustrates the syntax of XQuery trigger. The CREATE TRIGGER clause is used to define a new XQuery trigger, with the specified name. Rules can be prioritized in an absolute ordering, expressed with an optional WITH PRIORITY clause. The BEFORE/AFTER clause expresses the triggering time relative to the operation. Each trigger is associated with a set of update operations (insert, delete, rename, replace), adopted from the update exten-

sion of XQuery (Tatarinov et al. 2001). The operation is relative to elements that match an XPath expression (specified after the OF keyword). One or more predicates (XPath filters) are allowed in the steps to eliminate nodes that fail to satisfy given conditions. Once evaluated on document instances, the XPath expressions result into sequences of nodes, possibly belonging to different documents. The optional clause FOR EACH NODE/STATEMENT expresses the trigger granularity. A statement-level trigger executes once for each set of nodes extracted by evaluating the XPath expressions mentioned above, while a node-level trigger executes once for each of those nodes. DO clause express the action part that could be update operations or external operation such as send email or invoke SOAP procedural

```
CREATE TRIGGER CascadeDelete
AFTER Delete OF document("Portfolio.xml")/Holder
FOR EACH NODE
LET $OwnsOfDeletedHolder := (
          FOR $d IN document("Portfolio.xml")//Own
          WHERE $d[OHRegNo= OLD-NODE/Holder/HRegNo])
          RETURN $d )
WHEN ( not( empty($OwnsOfDeletedHolder )) )
DO ( FOR $Owns IN document(" Portfolio.xml")//Owns,
             $Own IN $Owns/Own[OHRegNo=$OwnsOfDeletedHolder/OHRegNo]
          UPDATE $Owns
          { DELETE $Own)
```

Figure 2.4: Rule 1 written using XQuery trigger.

Active XQuery Example Figure 2.4 illustrates Active XQuery trigger for *Rule 1* defined in Sub-Section 2.4.1. The trigger has a unique Name, *CascadeDelete*. This trigger fired after the deletion of a *Holder* element. *"document("Portfolio.xml")/Holder"* is an XPath expression that specifies the portion of the XML document that is to be monitored for the event *Delete*. *"FOR EACH NODE"* means the trigger granularity is node-oriented. *"LET $OwnsOfDeletedHolder"* is a let clause that defines variable named *OwnsOfDeletedHolder*. This variable is used in *where* clause. *Condition* is a Boolean predicate, *(not (empty ($OwnsOfDeletedHolder)))*, that specifies the condition under which the trigger is to be fired. The semantics of the condition means

the deleted holder has owns. *"Do"* is the action part of that trigger. It deletes the *owns* of the deleted holder.

2.4.2.2 ECA language for XML

Bailey et al. (2002b) have proposed ECA language for XML as a simple ECA rule language for providing reactive functionality on XML database. It uses the XQuery to construct new XML fragment, and uses XPath (Berglund et al. 2005) to determine a certain XML part. Several techniques for analyzing and optimizing this language were presented by Bailey et al. (2002a). Papamarkos et al. (2003) hav used the same concept and syntax of the ECA language for XML to extend RDF (Manola and Eric 2004) to support ECA rule paradigm as a tool for web semantics. A distributed system architecture for supporting ECA rules on distributed RDF repository is proposed by Papamarkos et al. (2003).

ECA language for XML Syntax. The syntax of ECA XML takes the following form:

on *event*
if *condition*
do *actions.*

The event part of an ECA rule is an expression of the form INSERT e or DELETE e. where e is a simple XPath expression which evaluates to a set of nodes. The rule is said to be triggered if this set of nodes includes any node in a new sub-document, in the case of an insertion, or in a deleted sub-document, in the case of a deletion. The condition part of an ECA rule is either the constant TRUE, or one or more simple XPath expressions connected by the boolean connectives *and, or,* and

not. If the condition references the system-defined *$delta* variable, it is evaluated once for each instantiation of *$delta* for each document. Otherwise, the condition is evaluated just once for each document. The actions part of an ECA rule is a sequence of one or more actions: *action 1; . . . ; action N*. These actions are executed on each XML document which has been changed by an event of the form specified in the rules event part and for which the rules condition query evaluates to *True*. The actions could be insert or delete operations. XQuery is used in these ECA rules only where there is a need to be able to construct new fragments of XML.

```
on DELETE document( Portfolio.xml )/Holder
if (document( Portfolio.xml )/Owns[OHRegNo=$delta/Holder/HRegNo])
do DELETE document( Portfolio.xml )/Owns[OHRegNo=$delta/Holder/HRegNo]
```

Figure 2.5: A trigger written by using ECA language for XML.

ECA language for XML Example Figure 2.5 illustrates a trigger writen using ECA language for XML to implement *Rule 1* defined in Sub-Section 2.4.1. In this example, the specified event that activates the trigger is delete a *holder* in the path *"document(Portfolio.xml)/Holder"*. The trigger's condition checks whether the deleted *holder* has *owns* or not. The action delete the *owns* that belong to the *holder*.

2.4.2.3 AXML

Abiteboul et al. (2002) have designed AXML for supporting data and service integration over the web. It combines XML documents with embedded calls to web services. The *Condition-Action* model rule, in which the action part could refer to some of free variables refereed to the condition part, was proposed by Sistla and Wolfson (1995). AXML uses the *Condition-Action* model.

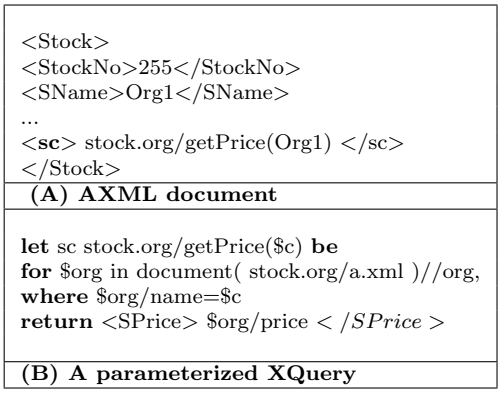

Figure 2.6: Rule 2 written using AXML.

AXML Syntax. Figure 2.6.A illustrates an AXML document that contains an element $< sc >$, service call, or more. These elements represent service calls that are embedded in the AXML document. The element $< sc >$ is activated according to specified time interval, when the AXML document is retrieved or queried, or whenever desired information are changed.

The called service could be expressed as parametrized XQuery, as illustrated in Figure 2.6.B. It could be used to evaluate a certain condition. The called service is stored detachedly from the AXML document. The called service returns its result as a sub-tree that is inserted as sibling elements of $< sc >$.

AXML Example. Figure 2.6 illustrates an AXML trigger for *Rule 2* defined in Sub-Section 2.4.1. This trigger is activated when the element $< sc >$ is retrieved or the AXML document is queried. The function *getPrice* represents the condition, which is the organization name equals to the name of the stock. The function returns the element *SPrice*. The action of AXML trigger is insertion to returned result as sibling elements of $< sc >$.

2.4.2.4 Active XML Schemas (AXS)

AXS, which has been developed by Schrefl and Bernauer (2001), reuses concepts from active database systems and event-based business rules to automatically and asynchronously manage the distributed Web content. It integrates passive and active behaviour into document schemas that can be stored and queried just as other document data. AXS was designed to support Web content management.

```
<xs:element name="rule">
<xs:complexType>
<xs:sequence>
<xs:element name="condition" type="actm:NativeCode" minOccurs="0"/>
<xs:element name="action" type="actm:NativeCode"/>
</xs:sequence>
<xs:attribute name="name" type="xs:QName" use="required"/>
<xs:attribute name="priority" type="xs:integer" use="optional"/>
<xs:attribute name="definedOn" type="xs:QName" use="required"/>
</xs:complexType>
</xs:element>
```

Figure 2.7: Active XML metaschema for rules.

AXS Syntax. The Active XML Metaschema for rules is shown in Figure 2.7. A rule is described by a name and priority. It is defined on an event class. Event class is a class, in which events are collected. Events are happenings of interest to a document. Each rule is defined upon an event class. If an event class occurs, all rules defined on that event class are triggered. A rule comprises a condition and an action. The condition and action are expressed using XSLT. *actm*, which is shown in Figure 2.7, is a namespace for Active XML Metaschema. The minimum occurrence of the element condition is 0. It means that the rule might be without a condition.

AXS Example. Figure 2.8.B shows the active document for *Rule 1* defined in Sub-Section 2.4.1. It defines the rule *RemoveOwns* on the event class *DeleteHolder*. An instance of this class is shown in Figuer 2.8.A. The rule condition tests whether

```
<DeleteHolder>
<actf:even id='e168' status='occured'>
<actf:occurrenceTime pt='2005-05-01...'/>
<actf:return xsi:nil='true'/>
<Holder>
<HRegNo>15</HRegNo>
</Holder>
<actf:even>
<DeleteHolder>
```

A

```
<rule definedoOn='DeleteHolder' name='RemoveOwns'>
<condition lang='http://www.w3.org/1999/XSL/Transform'>
<xsl:value-of select="$evt//Holder[HRegNo=document("Portfolio.xml")//OHRegNo]"/>
</condition>
<action lang='http://www.w3.org/1999/XSL/Transform'>
<invokeOperation name='removeOwns'>
<parameter name ='job'>
<xsl:value-of select='$cond'/>
</parameter>
</invokeOperation>
</action>
</rule>
```

B

Figure 2.8: (A) An instance of the event class *DeleteHolder*. (B) Rule 1 written using AXS.

the holder's has *owns* or not by querying the elements *HRegNo* and OHRegNo. If the condition applies, the holder's *owns* is removed from the *Portfolio.xml* by invoking operation *removeOwns(j:HolderRegNo)*.

2.4.2.5 Activeweb

Kiyomitsu et al. (2001) have developed Activeweb for supporting Web personalization, such that a Web page including its hyperlinks is changed according to each user's browsing history. Activeweb is used to provide XML-based active rules for deriving Web views and for defining access control. Activeweb aims at dealing with Web pages, html document, rather than dealing with XML document.

Activeweb Syntax. As shown in Figure 2.9, the Activeweb rule has three subelements *EVENT, CONDITION* and *ACTION*,and two attributes *ID* and *NAME*. An attribute *ID* indicates the identifier of the rule which is used by the system. The other attribute *NAME* indicates the name of the rule which is used by the author.

```
< ! ELEMENT RULES (RULE*)>
< ! ELEMENT RULE (EVENT, CONDITI ON?, ACTION)>
< ! ELEMENT EVENT (READ) >
< ! ELEMENT EVENT (READ) >
< ! ELEMENT CONDITION (USER?,AGGREGATE?) >
< ! ELEMENT ACTION (ADDI|HIDE|REPPLACE) >
< ! ATTLIST RULE ID #IMPLIED NAME (#PCDATA?)>
```

Figure 2.9: A brief XML DTD specification for the Activeweb rules.

```
MP, () : (times=0), () \
⇒ activate(MP, content="read-me-first.html);

MP, () : (period=6 months), ()\
⇒ activate(MP, content="read-me-first.html);
```

Figure 2.10: Rule 3 written using Activeweb trigger.

The event part of a rule is: the subelement of *EVENT* is *READ* only, because Web systems allow users only to issue *"http_get"* requests. It is supported to determine the location from and date at which the read request is issued respectively. The historical condition part of a rule *CONDITION* has two sub-elements. A sub-element *USER* is used to evaluate an access history of a user who gives rise to the event of the rule or triggers the rule. *AGGREGATE* element indicates aggregation from the access history data of the server (e.g. times of all access to page A). With the action part of a rule, an author can specify *ADD, HIDE* and *REPLACE* functions which enable reconfiguration of the combination of contents in the page.

Activeweb Example Figure 2.10 illustrates an Activeweb trigger for *Rule 3* defined in Sub-Section 2.4.1. This trigger is activated when a user accesses the main page MP. If the user accesses the MP page for the first time or has accessed it since more than 6 months ago, then the system adds the content of the *read me html page* to the MP page.

2.4.2.6 ARML

ARML, which has been developed by Cho et al. (2002), is an XML-based rule definition language, called Active Rule Markup Language (ARML). It aims at enabling event-based business rules, which were defined in various rule languages, to be shared and reused among different systems. ARML was designed for dealing with relational database. Therefore, it does not use XQuery or XPath.

```
<!ELEMENT rule (ruleDef, event , condition ,action , coupling, precedence, info)>
<!ELEMENT ruleDef (ruleName, table?, ruleSet?)>
<!ELEMENT event (eventName — algebra)?>
<!ELEMENT condition (methodCall — boolean — algebra)?>
<!ELEMENT action (methodCall+)>
<!ELEMENT coupling (ec? , ca?)>
<!ELEMENT precedence (after?, before?)>
<!ELEMENT info (designer*, description?, category?)>
```

Figure 2.11: The general DTD for the ARML syntax.

ARML Syntax. Figure 2.11 illustrates the DTD for the ARML syntax. In general, the rule consistes of the *ruleDef, event , condition ,action , coupling, precedence* and *info*. The *ruleDef* element defines the rule with a *rule name, rule set* and *rule table*. The *event* element defines the rule event. The *event* could be a composite event. The supported operators among the events are *"and", "or"* and *sequence*. The element *condition* might consist of one or more sub-elements *methodCall* that represents application-specific conditions. The supported operators are *"and", "or"* and *sequence*. The sub-element *methodCall* returns boolean value. The element *action* has series of sub-element *methodCalls* to describe simple or complex business logic. The element coupling has two sub-elements *ec* and *ca* that represent the transactional relationship among *event, condition,* and *action*. The element precedence defines the rule priority between rules triggered by the same event. The element *Info* specifies meta-information of the rule.

```
<rule>
<ruleDef> <ruleName>DoNotDel</ruleName>
<table> </table>Holder</ruleDef>
<event> <eventName> Delete </eventName> </event>
<condition> <methodCall> <methodName> holder.hasvalue </methodName>
<action> <methodCall> <methodName> holder.cancelDel </methodName>
          <methodCall> <methodName> holder.informeAdmin </methodName>
          <params> <param> <value> <int> Holder.HRegNO </int>
          </value> </param> </params> </methodCall> </action>
<info> <designer>escho </designer>
          <description> do not delete a holder that has value greater than 0 </description>
          <category>business</category> </info>
</rule>
```

Figure 2.12: Rule 4 written using ARML.

ARML Example Figure 2.12 illustrates the ARML trigger for *Rule 4* defined in Sub-Section 2.4.1. The trigger name is *DoNotDel*. ARML does not support XML document. It is assumed that *Holders* element is stored in a table called *Holder*. This trigger is activated when a holder is deleted. The *methodCall* element, whose value is *holder.hasvalue*, checks whether the deleted holder has value > 0 or not. If it is true, the *methodCall*, whose value is *holder.cancelDel*, cancels the delete operation and the *methodCall*, whose value is *holder.informeAdmin*, that takes the holder registration number as parameter to inform the system manager by that through an email.

2.4.3 The CoAX Framework Dimensions

This sub-section presents the dimensions of the CoAX framework that: 1) determines the fundamental characteristics of the XML-based ECA rule languages. Some of these features are inherited from active database, others relate to the nature of XML data and others cover the needs of real-world situations, such as temporal support for ECA rule paradigm; and 2) analyses and compares the XML-based ECA rule languages in order to determine the circumstances, in which the use of a certain language is more appropriate.

To the author's best knowledge, there was no framework for analysing the XML-

based ECA rule languages before CoAX. For short, the term active XML is used to refer to the XML-based ECA rule languages. The methodology, which has been used to build up CoAX, is based on:

- investigating into the structure of the active XML languages;
- determining how they execute the active behaviour;
- studying the management aspects provided by these languages to deal with the specified rules;
- determining the purpose of these languages;
- classifying the information targeted by the active rules according to its storage format;
- deciding whether these languages support the distributed environment or not;
- determining the implementation approaches and tools used in implementing these languages; and
- determining the features that should be provided in order to support real world active applications.

The objectives of the thesis focus on providing a formal language for the complex information and an implementation for this language using the available technologies, and applying this language to several application domains. Therefore, CoAX analyses the XML-based ECA rule languages in three dimensions: *knowledge*, *implementation* and *application*. The *knowledge* dimension focuses on the specification and execution model of these languages and the management aspects supported by them. The *implementation* dimension focuses on the utilized implementation methods and the distributed support. The *application* dimension focuses on the

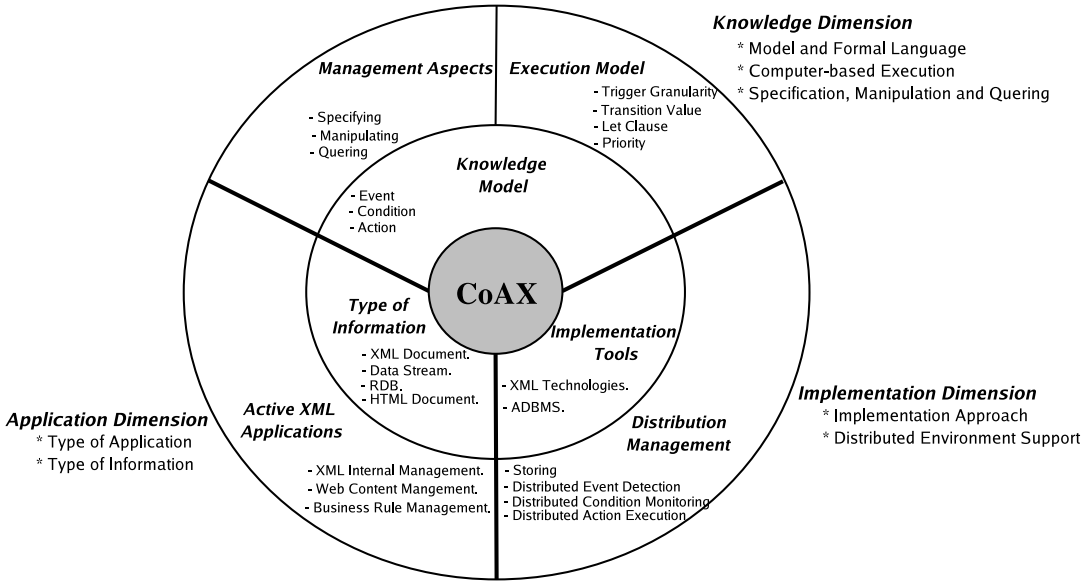

Figure 2.13: The CoAX framework.

application domains, to which the languages were applied, and information format supported by these languages.

Figure 2.13 illustrates the CoAX dimensions. The first dimension is *Knowledge Dimension* that has three categories:

- *knowledge model* that determines the main structure for the active rules. According to the *knowledge model*, the features for modelling a formal language are determined;
- *execution model* that determines how an active rule or more behave in the runtime; and
- *management model* that determines the required management functionality for active rules.

The second dimension is *Application Dimension* that has two categories:

- *type of application.* According to the application, the purpose and characteristics of the language are determined. The *type of application* is classified into

Web content management and *business rule management*; and

- *type of information.* The targeted information is classified according to its storage format into information stored in XML document, data stream, relational database and HTML document;

The third dimension is *Implementation Dimension* that has two categories:

- *implementation tools.* The tools used are ADBMS, the relational database systems provide support for active mechanism, and XML technologies; and
- *Distributed Management.* It might be needed that the active XML languages and systems provides management support for the active rule within distributed environment. That means the distributed event detection, condition monitoring and action execution should be supported over distributed environment. Moreover, it is needed to determine where the rule specification should be stored. In the following three sections, this dimensions are discussed in details.

2.4.4 Knowledge Dimension

This sub-section presents the details of the *knowledge dimension*. This dimension focuses on three required functionalities: *model and formal language*; *computer-based execution* and *management aspect*, for active rules. These functionalities were studied in the database area (Widom and Ceri 1996; Paton and Diaz 1999). However, it is new to consider them in a framework for the XML-based ECA Rule languages. In the following three subsections, these functionalities are discussed.

2.4.4.1 Knowledge Model

The *knowledge model* for active XML rules consists of three parts event, condition and action. Event-Condition-Action (ECA) rule paradigm refers also to the

knowledge model for active semantics. The semantics of ECA rule paradigm is that when an event E occurs, evaluate a condition C, and if the condition is satisfied, then execute an action A (Widom and Ceri 1996; Paton and Diaz 1999). Table 2.1 illustrates the features of each part. In the following, these features are discussed.

Knowledge Model Features	
1.1 Event	
Primitive	Insert Delete Update Retrive
Advanced	Time Composite and Temporal Application Defined External Defined
Event Granularity	
1.2 Condition	
Conventional Information	Predicate Query Method
Temporal Information	Temporal Predicate Temporal Query
1.3 Action	
Primitive	Insert Delete Update Retrive
Advanced	Time Composite and Temporal Application Defined

Table 2.1: Knowledge model aspects

2.4.4.2 Event

Event is an occurrence of significance to a task or action that happens inside or outside XML Repository and cause the ECA rule to be triggered. The features of the events are classified to primitive, advanced events and event granularity.

Primitive Events. Data manipulation operations, *insert, delete*, *update*, and *data retrieval* are considered as primitive events. *Insert* means to add a new XML element or sub-tree at certain position. *Delete* means to delete an XML element or a sub-tree. *Update* means to change the value or the content of an XML element or change the parent of an XML element or a sub-tree. Retrieving any part of an XML document might be considered as an event.

Advanced Events. In order to support the comprehensive rules that could be used to describe real world situations, advanced features should be added, such as composite and temporal, time, external and application defined events. A taxonomy for composite and temporal events has been proposed by Al-Kateb et al. (2003). Detecting the XML composite events has been studied by Bernauer et al. (2004).

- **Time Event.** A *time event* specifies that a rule should be triggered at certain time. That time may be absolute time, such as 18 March 05 at 10:30, or relative time, such as every month and last day in the week.

- **Composite and Temporal Event.** According to the taxonomy proposed by Al-Kateb et al. (2003), *Composite event* is a set of correlated events. These events could be primitive or composite events. The relationship between these events are logical or temporal relationship. Logical relationships are "*and*", "*or*" and "*not*". Temporal relationships are *Time Window* and *Preceding Relationship*. *Time Window* means more than one event happen within the same time period, such as events *E1* and *E2* happen within the same time period. *Preceding Relationship* means there are time order between the events, such as event *E1* happens before/after event *E2*. In this thesis, temporal event is used to refer to a composite event that has temporal relationship.

- **Application Defined Event.** *Application defined event* is an event E that is defied by the application, such that the application sends a signal that means the event E happen. Then all rules that specify E as an event are triggered. *Application defined event* provides ability to the application to define events that could not be defined using the primitive events or using XML repository functions, such as *patient admission* and *test result received*.

- **External Defined Event.** *External defined event* is an event E that define

external entity happing or occurrence, such that this entity sends a signal that means the event E happen. Then all rules that specify E as an event are triggered. The *external defined event* supports the interaction between distributed applications that collaborate to achieve a shared objective.

Event Granularity. The granularity of an event determines whether the event is defined for all entities, such as manipulation operations for all student, or for sub-entity, manipulation operations for fresh students (Paton 1999). It means adding conditions on the declaration of the event. This feature is important to the XML data. It could be supported using XPath that allocate a node in XML tree according to a filter expression.

2.4.4.3 Condition.

Once the rule is triggered, the specified conditions are evaluated to determine whether the action will be executed or not. The condition might be on conventional information or temporal information, which reflects the history and evolution of an object instance. The condition might be a predicate, a query, or a method (Widom and Ceri 1996; Bonifati 2001).

Non-Temporal Information. Non temporal information does not reflect the history. It is supported by the conventional query languages, such as SQL and XQuery. Conditions on the non temporal information might be:

- **Predicate.** The condition might be a predicate that filters information, retaining some object instances and discarding others. A predicate is defined using XML query languages. The result of the predicate is boolean value, true or false. The predicate might be simple or complex.
- **Query.** The condition might be a complete query that is encoded using an

XML query language. The condition is evaluated to be true, if the query returns non-empty result. Otherwise, the condition is evaluated to be false.

- **Methods.** The condition might be specified as a call to a procedure written in an application programming language or a call to a web service (He et al. 2004). The condition is evaluated to be true, if the method returns true. Otherwise, the condition is evaluated to be false. A method provides compensation for the shortcomings of the query languages and support to interaction between collaborative distributed applications.

Temporal Information. In order to support real world situation, the history and the evolution of object instances should be stored to provide support to rules that depend on the history of object instances. Time is an important aspect of all real-world application. The ability to model the time dimension is essential to many real-world applications, such as banking, inventory control, health-care, and geographical information systems (Goralwalla et al. 1995; Abraham and Roddick 1999; Terenziani et al. 2000). Conditions on temporal information might be:

- **Temporal Predicates.** The condition might be a temporal predicate that filters information according to temporal aspects. Temporal predicate key words are such as overlap, contain, preceding, last. Temporal predicates key words for Temporal Database are proposed by Snodgrass et al. (1994). Temporal XML predicates are proposed by Mansour (2003).

- **Temporal Query.** The condition might be a temporal XML query. Temporal XML query language supports temporal aspects such as coalescing and temporal predicates. Coalescing is a necessary operation that should be performed to arrange temporal data before executing other temporal operations (Zaniolo et al. 1997). Moreover, the other aspects of XML query, such as evaluation

path expression, should be extended to support the temporal dimension. Temporal XPath evaluation and temporal extensions for XPath have been studied by Mansour (2003).

2.4.4.4 Action

The action is executed when the rule is triggered and the condition is true. The action is classified into *primitive action*, *data manipulation*, *data retrieval*, and *advanced actions*, such as time and application defined actions.

Primitive Actions. The primitive actions might be a data manipulation operations, *insert, delete*, *update*, and *data retrieval*.

Advanced Actions. In order to support the comprehensive rules that could be used to describe real world situation, advanced actions might be needed, such as time, composite and temporal, and application defined actions. Events and Actions correlate to each other. Intuitively, an event causes an action of the evaluated rules, and an action may be cause an event.

- **Time Action.** Although, the rule is triggered and the condition is evaluated to true, however it might be needed to execute an action at certain time in the future. That time may be absolute time, such as first of June, or relative time, such as after the triggering time by 5 hours.
- **Composite and Temporal Action.** The rules, which have the same event and condition clause, could be rewritten as one rule with composite conjugation actions. Likewise, the rule might be specified set of actions. If these rules have partial order, priorities, among them, the composite conjugation actions might have a preceding among the actions, it called Temporal Actions. For example, execute action *A1* then after 5 hours execute *A2* and *A3*.

- **Application Defined Action.** The *application defined action* provides the application ability to define the actions that could not be defined as primitive actions, such as sending an email, SMS message or invoking an procedure or web services. Invoking a procedure or web services could be used to support the interaction among distributed applications that collaborate to achieve a shared objective.

2.4.5 Execution Model

The *execution model* determines how a set of rules is treated at run time (Widom and Ceri 1996; Paton 1999). Intuitively, the features of active database execution model should be restudied according to XML data model. According to XML data model, Bonifati (2001) studied some of these features, such as trigger granularity, transition values.

2. Execution Model	
Trigger Granularity	Set Node
Transition Value	New Old
Let Clause	
Priority	

Table 2.2: Execution model aspects

Trigger Granularity. The granularity determines the relationship between event occurrences and rule instance. The same event E might be simultaneously happened for different XML elements, then there are different event instances of E. If the collection of these event instances are used together to trigger a rule, that is called set-oriented granularity. However, If each single event instances is used to trigger a rule, that is called *node-oriented* granularity. It means if N nodes are affected by an operation, with *node-oriented*, N rule instances will be created, and with *set-oriented* only one rule instance will be created. Comparing with trigger features in

SQL3 (Kulkarni et al. 1999), a *node-oriented* trigger does not execute if the set of affected nodes is empty. However, a *set-oriented* trigger always executes once, even if the set of affected nodes is empty.

Transition Values. Both the trigger action and condition should be able to access the current state of the database, the *old* and the *new* values of the affected nodes. With *set-oriented*, the trigger condition or action may refer to the set of affected nodes by means of two transition set, *New-Set* and *Old-Set*. The *New-Set* contains the new values of the affected nodes. The *Old-Set* contains the old values of the affected nodes. With *node-oriented*, the trigger condition or action may refer to the affected node by means of two transition variables, *New* and *Old*. The *New* contains the new values of the affected node. The *Old* contains the old values of the affected node. Intuitively, with insert operation there is no old values, and with delete operation there is no new values. Therefore, with insert operation, the *Old* variable or *Old-Set* could not be used, and with delete operation, new variable or *New-Set* could not be used.

Let Clause. Let clause could be used to define a variable, whose scope covers the condition and the action (Bonifati 2001).

Priority. Priority is used to select a rule from a collection of the rules that is fired at the same time. Priority might be user defined or system defined. In user defined, user assigns integer number to a rule definition. In system defined, the system determines the selected rule according to system criteria, such as creation time of the rule.

2.4.6 Management Model

Active database and *active* XML specify and execute rules individually. However, the active part of the complex information should be represented as one unit of cohesive and correlative rules, which generally could be seen as a set of ECA rules and metadata. A cohesive and correlative event-based rules should be specified as correlated rules that have certain objectives and metadata. The objective of certain correlated rules R could be evaluated using a set of rules that triggered when the rules of R are executed, and they query the execution of the rules of R. The rules, which support the objective, are handled such as the other ECA rules. Evaluating the objective of the business rules provide support in modifying the business rules.

The execution of these business rules means the execution of each rule. The manipulation operations are one of the desired features of managing business rules. A rule might be *added, deleted* or *modified*. Likewise, a rule might be *activated* or *deactivated*. Querying the business rules is required to obtain information about certain unit of business rules and their execution. Querying the business rules and their execution provides ability to replay what happened during certain period. Temporal query could provide important information for the users.

2.4.7 Application Dimension

2.4.7.1 XML-based ECA Rule Applications

Active rules could be used to support different types of applications ranging from database internal applications to active behaviour that presents in many real-world domains (Zoumboulakis et al. 2004; Bailey et al. 2005; Bry et al. 2006). In this thesis, XML-based ECA Rule applications are classified into two categories, *Web Content Management* and *Business Rule Management*.

Web Content Management focuses on the applications of active rules that support classical XML management features, such as integrity constraints, web content management, replication management, temporal XML management, XML security management. The common features of the XML internal applications are that: 1) the rules of these applications are system-generated and hidden to the user; and 2) the generation process is fully automated.

Business Rule Management focuses on real world applications that need to react to events which occur in the real world with tangible side effects on the database contents. Business rules normally would be encapsulated in an application code. One class of the business rules is alerters. The action of alerters rules consists of sending message, such as Short Message Service (SMS).

2.4.7.2 Type of Information

The XML-based ECA rules might be targeted different kind of data, on which conditions are evaluated and/or actions are performed. According to the applications, the data source is determined. Intuitively, the kind of data source determines the technologies that might be used to implement the XML-based ECA rules language.

XML repository might be a native XML repository or a relation database repository, which is used store XML document using database schema, such as (Florescu and Donald 1999; Yoshikawa and Amagasa 2001), by converting from XML tree structure to relational structure. XML document might be an XML View that could be used to publish relational database. Business rules might be defined over these XML views. The trigger implementation of these rules could be supported by SQL.

Data Stream is a continuous, rapid, real-time and ordered flow of data or sequence of items (Golab and Tamer 2003; Babcock et al. 2002). Data stream is

suitable when the data is changing constantly and there is no need to operate on large portions of the data multiple times (Babu and Widom 2001). Some XML-based ECA rule languages use XML format and aim at supporting active semantics over relational database or Web pages.

2.4.8 Implementation Dimension

In order to store the rule specification within distributed environment, there are different alternatives to determine where the specification should be stored. One of the challenge for managing the ECA rules within distributed environment is detecting the events that happen in remote external locations with consideration to the differences in time and a possibility of a delay.

The condition might be access information that is not stored in the local site. It is needed to provide distributed monitoring for the conditions. Within distributed environment the actions could take place in an external entities

2.4.9 Implementation Approach

The relational database technologies is used to store and query XML database, such as (Florescu and Donald 1999; Yoshikawa and Amagasa 2001; DeHaan et al. 2003; Grust et al. 2004). Active database could be used in implementing XML-based ECA Rule languages. XML technologies, such as DOM (Hors et al. 2000), could be used to implement the XML-based ECA rule language.

XML technologies, such as XQuery and XML schema, might be integrated with XML-based ECA Rule languages. These technologies could be used in one or more of active XML paradigm, event, condition and action. That integration increase the functionality and capabilities of the active XML languages. Web services could be invoke by condition or action part of an active XML language. Web services

could be used to support interaction among distributed application.

2.4.10 Comparing the XML-based ECA Rule Languages Using CoAX

The XML-based ECA rule languages are evaluated according to the CoAX framework. Table 2.12 illustrates the summary of this evaluation. When the language fully supports a certain feature, it is denoted as *Yes*. When a feature is not completely covered, it is denoted as *Partially*. Otherwise, it is denoted as *No*.

2.4.10.1 Knowledge Model

Event Active XQuery supports *primitive events*, *insert, delete* and *update*. However, it does not support the *retrieving events* and the *advanced events*. It supports the *event granularity* because it uses XPath expression to determine the nodes that are affected by a certain event.

ECA language for XML supports *primitive events*, *insert* and *delete*. However, it does not support the *update* and *retrieving events* and the *advanced events*. It supports the *event granularity* because it uses XPath expression to determine the nodes that are affected by a certain event.

AXML implements the rules, which are used to support data integration, by using web services. These web services are triggered according to specified time interval, when AXML document is retrieved or queried, or whenever desired information are changed (Abiteboul et al. 2002). In order to support the later one, AXML use the extension to XQuery proposed in (Tatarinov et al. 2001). AXML supports the primitive events, *insert, delete, update*, *retrieve*, and *time-based* events. It does not support the *event granularity*.

In Activeweb, the event clause represents only a user's behaviour that could be reading or accessing a web page from another one, on specific location, and in certain

date. The events of Activeweb are application defied events, such as requesting for reading a page from location A in time t.

Active XML Schemas supports different event types, *primitive events*, *composite events* and *application defined events*. The *event granularity*, however, is not supported. ARML supports *primitive events*, *insert, delete* and *update*. However, it does not support the *retrieving events*. It also supports *composite events*. However, it does support the other *advanced events* and the *event granularity*. Table 2.3 shows the summary of the event support in the selected languages.

Active XML Features		Active XQuery	ECA XML	AXML	Activeweb	AXS	ARML
A.1. Knowledge Model							
A.1.1 Event							
Primitive	Insert	Yes	Yes	Yes	No	Yes	Yes
	Delete	Yes	Yes	Yes	No	Yes	Yes
	Update	Yes	No	Yes	No	Yes	Yes
	Retrive	No	No	Yes	No	Yes	No
Advanced	Time	No	No	Yes	No	Yes	No
	Composite and Temporal	No	No	No	No	Yes	Yes
	Application Defined	No	No	No	Yes	Yes	No
	External Defined	No	No	No	No	No	No
Event Granularity		Yes	Yes	No	No	No	No

Table 2.3: Event features.

Condition. Active XQuery supports the conditions on *snapshot information* as a predicate and a query. However, the method and the condition on the *temporal information* are not supported. In ECA language for XML, the condition might be one or more simple predicates, XPath expressions, on *snapshot information*. However, the query, method and also the condition on the *temporal information* are not supported.

AXML supports the conditions on *snapshot information* as a predicate and a query. The conditions could be encoded using XPath or XQuery. The conditions might be sent to the web service as parameters. In Activeweb, the condition clause deals only with user's access history and aggregated information of all user's access history. User's access history only consider the time at which the user accesses the page, the period from when the user accessed last, or the pages and the links that

the user has already accessed or navigated. Intuitively, the historical conditions in Activeweb are partially supported for temporal predicate and temporal query.

Active XML Schemas expresses the condition using XSLT. Invoking a method is not supported. ARML supports the conditions on the *snapshot information* as a predicate, a query and a method. However, conditions on the *temporal information* are not supported. Table 2.4 shows the summary of the condition support in the selected languages.

Active XML Features		Active XQuery	ECA XML	AXML	Activeweb	AXS	ARML
A.1. Knowledge Model							
A.1.2 Condition							
Non-Temporal Information	Predicate	Yes	Yes	Yes	No	No	Yes
	Query	Yes	Yes	Yes	No	Yes	Yes
	Method	No	No	No	No	No	Yes
Temporal Information	Temporal Predicate	No	No	No	Partially	No	No
	Temporal Query	No	No	No	Partially	No	No

Table 2.4: Condition feature.

Action Active XQuery supports *primitive actions*, *insert, delete, update*, and *retrieve*. Moreover, it supports *application defined actions*. However, it does not support *composite and temporal* actions. In ECA language for XML, there are two kind of actions *insert* and *delete* actions. However, it does not support *update*, *retrieve* and *advanced actions*, such as *composite and temporal*, and *application defined* actions.

AXML inserts the sub-tree, which is generated by the triggered web services, into the AXML document. However, the other primitive actions, such as *delete* or *update*, and the *advanced actions* are not supported. With the action part of the Activeweb, an application defined actions, such as add, hide and replace functions, are applied for the contents in the page. The action part might perfore more functions. Activeweb supports application defined actions and partially supports the *composite and temporal* actions.

Active XML Schemas supports the *primitive* actions and *application defined actions*. ARML supports *primitive actions*, *insert, delete, update,* and *retrieve*.

Moreover, it supports *application defined actions*. However, it does not support *composite and temporal* actions. Table 2.5 shows the summary of the action support in the selected languages.

Active XML Features		Active XQuery	ECA XML	AXML	Activeweb	AXS	ARML
A.1. Knowledge Model							
A.1.3 Action							
Primitive	Insert	Yes	Yes	Yes	No	Yes	Yes
	Delete	Yes	Yes	No	No	Yes	Yes
	Update	Yes	No	No	No	Yes	Yes
	Retrive	Yes	No	No	No	No	Yes
Advanced	Time	No	No	No	NO	No	No
	Composite and Temporal	No	No	No	Partially	No	No
	Application Defined	Yes	No	No	Yes	Yes	Yes

Table 2.5: Action features.

2.4.10.2 Execution Model

Active XQuery supports the different types of the *trigger granularity*, *set-oriented* and *node-oriented*. ECA language for XML, AXML, Active XML Schemas and ARML do not consider the trigger granularity features.

Active XQuery supports the different types of the *transition value*. With *node-oriented*, the variable *OldNode* and *NewNode* denote the affected XML element in its before and after state. With *set-oriented*, the variable *OldNode* and *NewNode* denote the affected sequence of XML elements in their before and after state. ECA language for XML, AXML, Activeweb and Active XML Schemas do not consider the *transition values* feature. ARML supports the different types of the *transition value*, *new* and *old*.

In Active XQuery, XQuery variable could be defined. The scope of this variable covers the condition and the action clauses. ECA language for XML, AXML, Activeweb, Active XML Schemas and ARML do not consider the *let clause* feature.

In Active XQuery, the user can assign a signed integer number as *priority* for the rules. ECA language for XML does not provide the *user defined priority*. It assumes that no two rules can have the same *priority*. AXML, Activeweb and Active

XML Schemas do not support the *priority*. ARML provides ability to determine the precedence between the rules. Table 2.6 shows the summary of the *execution model* support in the selected languages.

Active XML Features		Active XQuery	ECA XML	AXML	Activeweb	AXS	ARML
A.1. Knowledge Model							
A.2. Execution Model							
Trigger Granularity	Set	Yes	No	No	No	No	No
	Node	Yes	No	No	No	No	No
Transition Value	New	Yes	No	No	No	No	Yes
	Old	Yes	No	No	No	No	Yes
Let Clause		Yes	No	No	No	No	No
Priority		Yes	Partially	No	No	No	Yes

Table 2.6: The execution model.

2.4.10.3 Management Model

The features of managing business rules are not fully supported by any of the proposed languages. However, AXML document are syntactically valid XML document. It could be stored, manipulated and queried using existing tools for XML. However, the AXML document includes only the call for web services. The web services are stored separately. Therefore, AXML partially supports the query and manipulation for rules, which are implemented using web services. Only the calls for the web services could be queried or manipulated.

Moreover, Activeweb and ARML rules are expressed in XML format that is validated by using a DTD. The existing tools for XML could be used to query and manipulate these rules. However, querying and manipulating ECA rules require extra features that are directed to deal with the rules. Table 2.7 shows the summary of the *management model* support in the selected languages.

Active XML Features	Active XQuery	ECA XML	AXML	Activeweb	AXS	ARML
A.3. Management Model						
Specifying	No	No	No	No	No	No
Manipulating	No	No	Partially	Partially	Partially	Partially
Quering	No	No	Partially	Partially	Partially	Partially

Table 2.7: Managment aspects.

2.4.10.4 XML-based ECA Rule Applications

Active XQuery and ECA language for XML focuses on the applications of *web content management* and could be partially used to implement the applications of the *business rule management*. AXML mainly supports data integration, which is a kind of *web content management* application. Activeweb mainly supports web personalization, which is a kind of *web content management*. Activeweb mainly supports applications of the *business rule management*. Active XML Schemas mainly supports *web content management*. Table 2.8 shows the summary of the type of the active XML applications support in the selected languages.

Active XML Features	Active XQuery	ECA XML	AXML	Activeweb	AXS	ARML
B.1. Active XML Applications						
Web Content Management	Yes	Yes	Yes	Partially	Yes	No
Business Rule Management	Partially	Partially	No	No	NO	Yes

Table 2.8: Active XML applications

2.4.10.5 Type of Information

Active XQuery could be used with XML document and XML views over relational database, as shown by Shao et al. (2004). ECA language for XML and AXML deals with XML document. Activeweb targets the html documents and Web pages. Activeweb targets the relational database. Active XML Schemas targets the XML document. Dealing with XML data stream is not studied. Table 2.9 shows the summary of the type of the information storage support in the selected languages.

Active XML Features	Active XQuery	ECA XML	AXML	Activeweb	Active XML Schema	ARML
B.2 Type of Information						
XML Document	Yes	Yes	Yes	No	Yes	No
Data Stream	No	No	No	No	No	No
Relational DB	No	No	No	Yes	No	Yes
HTML Document	No	No	No	Yes	No	Yes

Table 2.9: Type of information

2.4.10.6 Distributed Management Issues

Although AXS only discussed the distributed management issues, it studied only the *distributed event detection* and *storing the specification*. AXS considers the variation in time from site to site and also considers the delay that might happen in sending the notification. In order to solve that, AXS attaches with the *event class instance* three pieces of information: 1) the publication time, at which the remote document publishes the event, 2) the delivery time, at which the event is delivered, and 3) the occurrence time, at which the event is stored in the document. Using these timestamps, a partial order can be established. The rule administrator determines where AXS rules specifications should be stored. Table 2.10 shows the summary of distributed management support.

Active XML Features	Active XQuery	ECA XML	AXML	Activeweb	AXS	ARML
C.1. Distributed Management						
Storing	No	No	No	No	Yes	No
Event Detection	No	No	No	No	Yes	No
Condition Monitoring	No	No	No	No	No	No
Action Execution	No	No	No	No	No	No

Table 2.10: Distributed management

2.4.10.7 Implementation Approach

RDBM and XML technologies could be used to implement these languages. Active XQuery supports the XPath, XQuery, and Web Services. ECA language for XML is integrated with XPath and XQuery. AXML supports the XPath, XQuery, and Web Services. Activeweb partially supports XLST and XML Schema, DTD. Active XML Schemas supports XLST and XML Schema. ARML does not support any of the XML technologies except DTD and web services. Table 2.11 shows the summary of the implementation tools used in the selected languages.

Active XML Features		Active XQuery	ECA XML	AXML	Activeweb	AXS	ARML
C.2. Implementation Tools							
Using RDBMS		Yes	No	No	No	No	Yes
XML Technologies	XQuery	Yes	Partially	Yes	No	No	No
	XML Schema	No	No	No	Partially	Yes	Partially
	Web Services	Yes	No	Yes	No	No	Yes

Table 2.11: Implementaion tools

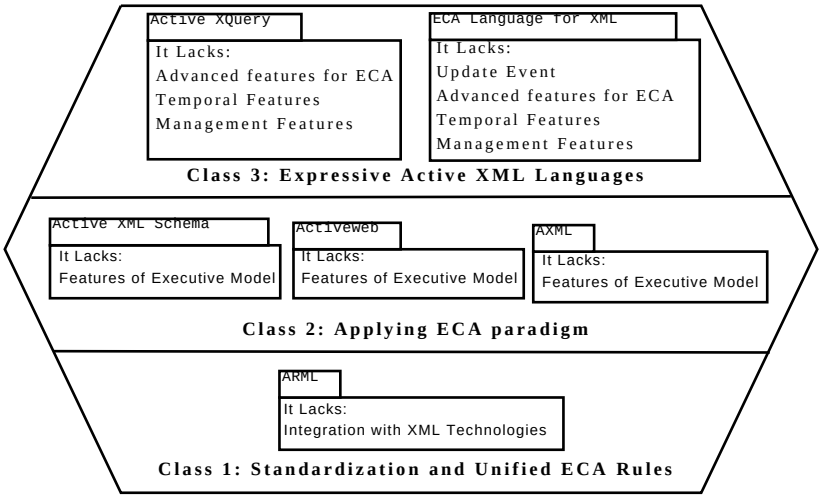

Figure 2.14: The taxonomy of the XML-based ECA rule languages.

2.4.11 A Taxonomy for the XML-based ECA Rule languages

The XML-based ECA Rule languages are classified to three classes, As illustrated in figure 2.14. The classification is made according to the number of the covered features, partially and not covered features.

Class 1: Standardized and Unified ECA rules. This category standardizes and unifies the ECA rule specification. The languages in this category do not deal with XML data. However, they use XML format to represent the rules.

Class 2: Applying ECA Rule Paradigm. This category focuses on implementing active XML solution to a certain problem rather than providing a complete language that could be used to express ECA rules over XML database.

Class 3: Expressive XML-based ECA Rule languages. This category includes languages that play the same role as high level SQL trigger standard and languages in relational database. Active XQuery provides the most features that are provided for SQL trigger in (Kulkarni et al. 1999).

2.5 Chapter Summary

This chapter has discussed shortcomings of the management provided to the complex information. The most related research areas to this thesis research are the workflow and computerised clinical guidelines. Both areas overlook the management of the complex information as a distinct entity, which consists of *active* and *passive* parts. The *active* part determines the recommended procedure that should be taken in specific situations. The *passive* part determines the information that describes these situations and other descriptive information.

This chapter has also classified the computerised clinical guidelines approaches adopting the ECA rule paradigm and XML into 5 categories: 1) a category supports clinical guidelines dissemination using XML; 2) a category utilizes the ECA rule paradigm to provide a specification and execution support to the clinical guidelines; 3) a category represents clinical events using XML; 4) a category supports the clinical events with the ECA rule paradigm; and 5) a category incorporates the XML into ECA rule paradigm to support the clinical guidelines management. All the approaches related to these categories focus on supporting the active part of the complex information. These approaches overlook the need to specify and manage the patient plan (complex information), which is produced by applying a specific clinical guidelines to a particular patient. The patient plan is seen in the healthcare domain, as one distinct entity.

In this chapter, the workflow approaches have been classified into three cat-

egories, Business Process Management (BPM), Adaptive Workflow and Process Mining. The process mining category is ignored because it is not strongly related to this thesis. In the BPM, the focus of the workflow approaches is to model and manage only the active part of the complex information as business processes. Several languages for modelling these business processes have been proposed. These languages are classified into graph and rule based languages. Most of the rule based languages are supported using the ECA rule paradigm. The adaptive workflow approaches focus on exception handling and logical failures during workflow execution. Instead, the complex information adaptation is to adapt the general specification, which represents the domain knowledge, to a specific domain entity, such as patient, before the execution.

This thesis work distinguishes from all these approaches by providing a generic approach and framework for managing the complex information at a platform-independent, application domain, and high level under a unified management environment. The complex information are to be managed under a unified framework that provides support to specify and formalize the complex information at a generic level (skeletal plan), instantiate complex information instances, such as a patient plan, using the formalized skeletal plan, execute these instances, keep the execution history incorporated into each instance, manipulate and query all these pieces of information at a high and declarative level.

A combination of ECA rule paradigm, XML technologies, and database systems is adopted as a method for realizing the author's approach and framework for managing the complex information. Therefore, the Author has developed a comparative framework, called CoAX. The CoAX framework considered the requirements demanded to support the complex information management to analyse the XML-based ECA rule languages in details. This analysis aimed at determining the

compatibility of these languages with the requirements of the complex information management and shortcomings of these languages.

The main findings of CoAX are that the ECA rule paradigm has been incorporated into XML to provide: 1) support to Web content management, such as in (Schrefl and Bernauer 2001; Abiteboul et al. 2002); 2) active behaviour support over XML data, such as in (Bonifati et al. 2002; Bailey et al. 2002b); and 3) support for sharing business rules among relational database, such as in (Cho et al. 2002). These languages have several weaknesses, such as 1) they are not at a user domain and high level; 2) they have lack of support to the temporal features required to store and retrieve the execution history of the complex information; and 3) they have lack of manipulation and query support.

A. Knowledge Dimension							
Features		**A-XQuery**	**ECA XML**	**AXML**	**Activeweb**	**AXS**	**ARML**
A.1. Knowledge Model							
A.1.1 Event							
Primitive	Insert	Yes	Yes	Yes	No	Yes	Yes
	Delete	Yes	Yes	Yes	No	Yes	Yes
	Update	Yes	No	Yes	No	Yes	Yes
	Retrive	No	No	Yes	No	Yes	No
Advanced	Time	No	No	Yes	No	Yes	No
	Temporal	No	No	No	No	Yes	Yes
	Application Defined	No	No	No	Yes	Yes	No
	External Defined	No	No	No	No	No	No
Event Granularity		Yes	Yes	No	No	No	No
A.1.2 Condition							
Non-Temporal Information	Predicate	Yes	Yes	Yes	No	No	Yes
	Query	Yes	Yes	Yes	No	Yes	Yes
	Method	No	No	No	No	No	Yes
Temporal Information	Temporal Predicate	No	No	No	Part	No	No
	Temporal Query	No	No	No	Part	No	No
A.1.3 Action							
Primitive	Insert	Yes	Yes	Yes	No	Yes	Yes
	Delete	Yes	Yes	No	No	Yes	Yes
	Update	Yes	No	No	No	Yes	Yes
	Retrive	Yes	No	No	No	No	Yes
Advanced	Time	No	No	No	NO	No	No
	Temporal	No	No	No	Part	No	No
	Application Defined	Yes	No	No	Yes	Yes	Yes
A.2. Execution Model							
Trigger Granularity	Set	Yes	No	No	No	No	No
	Node	Yes	No	No	No	No	No
Transition Value	New	Yes	No	No	No	No	Yes
	Old	Yes	No	No	No	No	Yes
Let Clause		Yes	No	No	No	No	No
Priority		Yes	Part	No	No	No	Yes
A.3. Management Model							
Specifying		No	No	No	No	No	No
Manipulating		No	No	Part	Part	Part	Part
Quering		No	No	Part	Part	Part	Part
B. Application Dimension							
Features		**A-XQuery**	**ECA XML**	**AXML**	**Activeweb**	**AXS**	**ARML**
B.1. Active XML Applications							
Web Content Management		Yes	Yes	Yes	Part	Yes	No
Business Rule Management		Part	Part	No	No	NO	Yes
B.2 Type of Information							
XML Document		Yes	Yes	Yes	No	Yes	No
Data Stream		No	No	No	No	No	No
Relational DB		No	No	No	Yes	No	Yes
HTML Document		No	No	No	Yes	No	Yes
C. Implementation Dimension							
Features		**A-XQuery**	**ECA XML**	**AXML**	**Activeweb**	**AXS**	**ARML**
C.1. Distributed Management for							
Storing		Yes	Yes	Yes	No	Yes	No
Event Detection		No	No	No	No	No	No
Condition Monitoring		No	No	No	Yes	No	Yes
Action Execution		No	No	No	Yes	No	Yes
C.2. Implementation Tools							
Using RDBMS		Yes	No	No	No	No	Yes
XML Technologies	XQuery	Yes	Part	Yes	No	No	No
	XML Schema	No	No	No	Part	Yes	Part
	Web Services	Yes	No	Yes	No	No	Yes
A-XQuery* is a short version of *Active XQuery* and *Part* is a short version of *Partially							

Table 2.12: Comparison of the XML-based ECA rule languages using CoAX.

3

SIM: A Generic Approach and Framework for Computerising the Complex Information

This chapter presents a generic approach and framework, called SIM, for managing the complex information and a method based on Temporal Active XML database for realizing the proposed approach and framework.

3.1 An Overview of the SIM Approach and Framework

This section presents an overview of the SIM approach and framework for the complex information management. SIM stands for Specification, Instantiation, and Maintenance of the complex information. In SIM, the complex information management is achieved through modelling the complex information as one distinct entity with different abstraction levels and managing this entity with multi-dimensional management, as depicted in Figure 3.1.

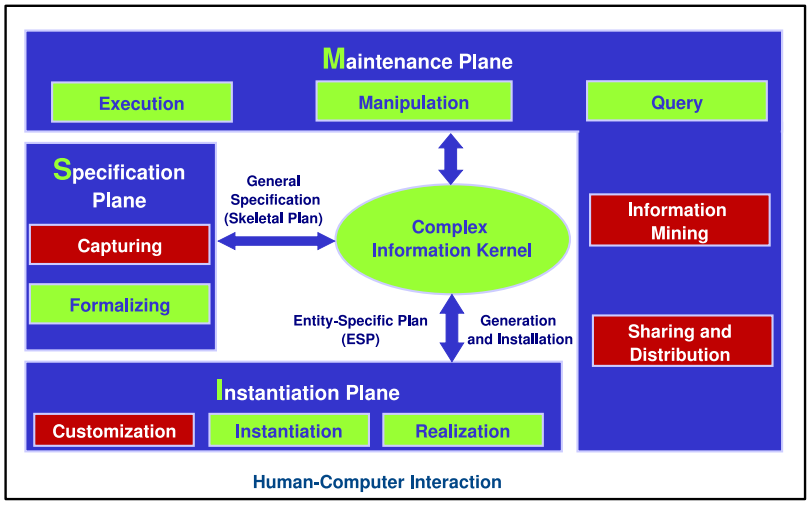

Figure 3.1: SIM: A generic approach and framework for computerising the Complex Information.

The SIM approach aims at modelling the complex information as one distinct entity, which is represented as plan that combines the active and passive information. This plan has a general specification (skeletal plan), from which an entity-specific plan is generated. The SIM approach models the complex information at different abstraction levels. For example in the cancer tumor disease management, the SIM approach could be used to produce general medical plans (skeletal plans), from which a patient plan (entity-specific plan) will be generated to suit a particular patient. These both kind of plans represent complex information at different abstraction levels, and should be maintained over the time as the patient state is changing.

The SIM framework aims at managing the complex information through three dimensions: *specification*, *instantiation* and *maintenance*. *Specification* is the formal definition of the complex information at a generic level (skeletal plan). *Instantiation* is a refinement for a specific skeletal plan to suite a particular entity. *Maintenance* is the work done to keep the complex information in a proper condition, which means:

- Execution. The complex information is executed as soon as a change of interest happens;
- Manipulation. The complex information is subject to the same manipulation operations, as other information, plus special operations, such as activate, deactivate, terminate and fire. Through the execution and manipulation, the execution history is logged in the complex information itself as an object growing over time;
- Query. As other information, the complex information is subject to the same queries plus special queries for recovering the complex information at any time point and review the complex information evolution over a time period;
- Information Mining. Analysing and comparing the complex information is to produce new, better and enhanced domain knowledge. That leads to better skeletal plans for a particular activity; and
- Sharing and Distribution. Most of the modern application domains have a distributed architecture, thus leading to domain users demanding the support for remote management for the complex information, which represents a successful practice in a specific situation for a specific entity.

A Human-Computer Interaction support is a common base for the three planes of the framework. The user interface to the three planes should be based on understanding the relationships among users' goals and objectives, their personal capabilities, the social environment, and the designed artifacts with which they interact. Human-Computer Interaction provides bi-directional support between the users and the system in order to support the different abstraction levels of the complex information management.

3.2 The SIM Approach to Modelling the Complex Information

This section presents the SIM approach that provides a conceptual model for the complex information and its life-cycle.

3.2.1 The Skeletal Plan and Entity-Specific Plan

The SIM approach aims at providing a conceptual model for the complex information at an application domain level with different abstraction levels. The complex information is classified into skeletal plan and entity-specific plan, as illustrated in Figure 3.2.

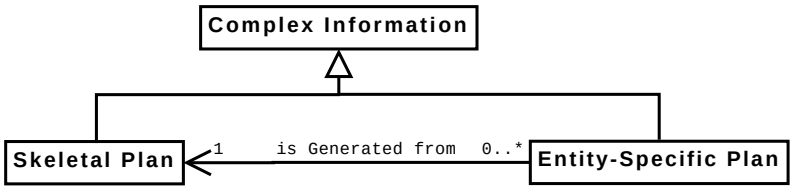

Figure 3.2: A class diagram for the relationship between the skeletal plan and the entity-specific plan.

The skeletal plan is a conceptual framework or model for the complex information at a generic level. As a logical framework, the skeletal plan defines the relationship between it's information members. The skeletal plan changes when necessary in order to be suitable for a particular organization and/or environment. The entity-specific plan is an instance plan generated from a skeletal plan for a particular entity. consequently, the entity-specific plan represents a real case for a particular skeletal plan. Table 3.1 provides a comparison between the skeletal plan and entity-specific plan.

The entity-specific plan is a conceptual model for the complex information at an entity-specific level. An entity-specific plan realizes its behaviour and state from its skeletal plan. The skeletal plan is static in the sense that it is almost fixed before,

Skeletal Plan	Entity-Specific Plan
static.	dynamic
contains one or more entity-specific plans	belongs to only one Skeletal plan
unlimited lifespan	limited lifespan

Table 3.1: A comparison between the skeletal plan and entity-specific plan.

during, and after the execution and does not have a state transition. However, the entity-specific plan is dynamic in the sense that it may undergo significant changes during the execution and it does have state transitions. An entity-specific plan should belong to only one skeletal plan. However, the skeletal plan might contain many entity-specific plans. An entity-specific plan has a limited lifespan. Entity-specific plans during their lifespan are created and eventually completed, terminated and/or suspended.

The SIM approach emphasizes the organisation of the complex information as one distinct entity. The complex information is

- a skeletal plan that 1) represents a general structure specified according to domain knowledge; and 2) deals with a particular activity; or
- an entity-specific plan that is 1) generated from a skeletal plan according to the users preferences and interest; 2) executed, as soon as a change of interest happens to the domain information; 3) a real case study of applying a particular skeletal plan.

The complex information, as one distinct entity, is subject to 1) manipulation and query as a first class object, not only as row data; 2) a distributed management that supports the remote users and distributed applications.

3.2.2 A Conceptual Model for the Complex Information

The conceptual model of the complex information is an abstract construct that represents the complex information, with a set of information components and a set of logical relationships between these components, as depicted in Figures 3.3. The model in this sense is constructed to organise of the complex information as one distinct entity. The complex information consists of *active* part and *passive* part. The *active* part represents the way in which an activity should behave and react in a particular situation. The information component under this part is expressing actions rather than states of being. The *passive* part is subject to changes without taking any action.

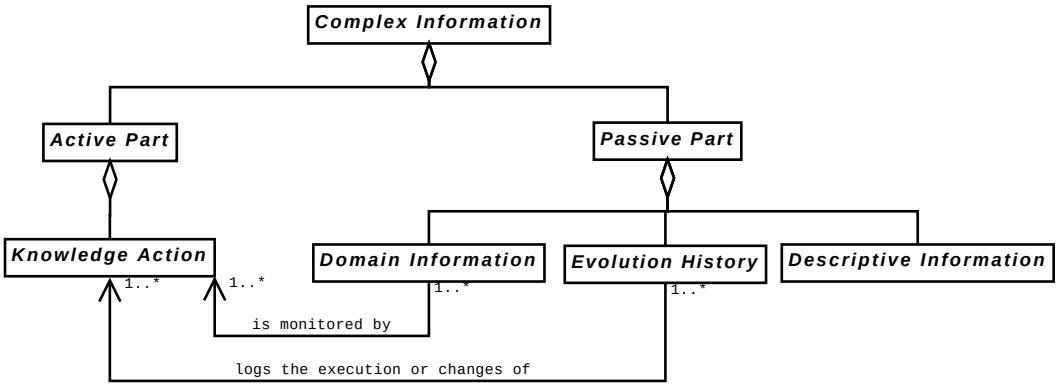

Figure 3.3: An UML class diagram for the complex information conceptual model.

Figure 3.3 illustrates an UML class diagram model for the complex information according to the nature of its information components. The *active* part is modelled by the *knowledge action* component that determines the reaction that should be taken as a response to a specific situation. The initial step for modelling the *active* part is to describe the primitive reactive decision logic for a specific situation. E.g. if a service is unavailable (event) and it is not maintenance time (condition) then send a notification (action). This primitive reactive decision logic is defined as an Event-Condition-Action rule. Therefore, the *knowledge action* component constitutes the

activity reactive behaviour as modularized sets of ECA rules.

The *passive* part is expressing states of being rather than actions. Its components are *domain information, evolution history*, and *descriptive information*. The *domain information* component models the situations, to which the *knowledge action* component reacts. Situations are represented through terms, whose value are monitored by the *knowledge action* component, e.g. if term "service" becomes unavailable, the rest of the rule will be evaluated. The *evolution history* component tracks changes to the initial complex information, dependencies, and goals, e.g. the primitive reactive decision logic might be changed over time. Moreover, the execution of the primitive reactive decision logic is also logged by the *evolution history* component. The *descriptive information* component provides didactic information, such as purpose, explanation, keywords, citation, and links, and release information, such as version, institution, author, and specialist.

Components	**Skeletal Plan**	**Entity-Specific (ES) Plan**
Knowledge Action	platform-independent	platform-specific
Domain Information	domain-specific and entity-independent	computer-specific and entity-dependant
Descriptive Information	specification-oriented	execution-oriented
Evolution History	logs modification	logs modification and execution

Table 3.2: A comparison between the complex information components.

The four components of the complex information, *knowledge action, domain information, evolution history*, and *descriptive information*, exist in both the skeletal plan and entity-specific plan, but at different abstraction levels, as depicted in Table 3.2. In the sense that the entity-specific plan is generated from a skeletal plan, the different abstraction levels are appeared.

The *knowledge action* component in the skeletal plan is a platform-independent, which means the rules should be formalized as platform-independent statements that could be directly mapped into executable statements of a software system.

However, the *knowledge action* component in the entity-specific plan is a platform-specific, which means the rules are statements in a language of a specific execution environment.

The *domain information* component in the skeletal plan is a domain-specific, which means the terms representing specific situations are defined using the domain terminologies. In the entity-specific plan, these terms should be mapped into computer interpretable terms, such as data items of a database schema. Moreover, in the skeletal plan the *domain information* component refers to attributes without values, but in the entity-specific plan, the *domain information* component refers to attributes with values.

The *descriptive information* component in the skeletal plan is a specification-oriented to provide a descriptive information regarding the specification and formalization process, such as information about the author. However, in the entity-specific plan, it provides a descriptive information related to the execution, such as person in charge of the ES plan, and a specific entity, to which the ES plan is generated. The *evolution history* component, in the skeletal plan, logs the modification made to the skeletal plan. However, in the entity-specific plan, the modification made to ES plan's components and the execution history of the *knowledge action* component.

3.2.3 The Complex Information Life-Cycle

In SIM approach, the complex information is either a skeletal plan; which is static in the sense that it does not have a state transition; or an entity-specific (ES) plan; which is dynamic in the sense that it has state transitions. Consequently, this section focuses on the state transitions of the ES plan, as shown in Figure 3.4.

The state transitions of the ES plan are predefined and context-sensitive. The

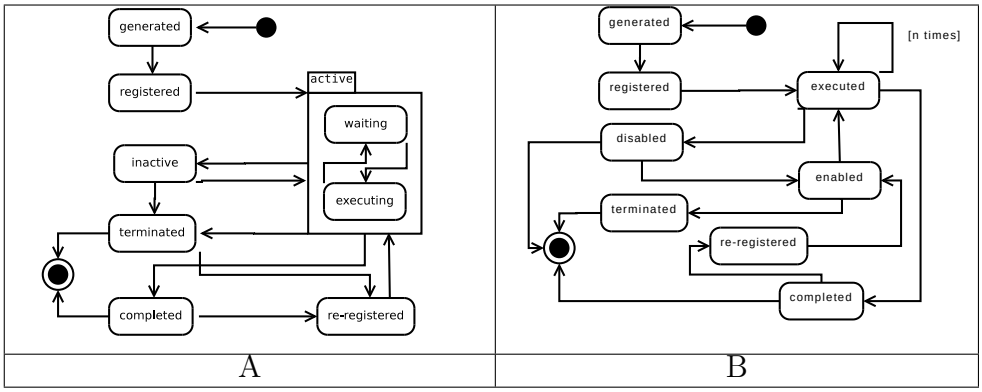

Figure 3.4: the life-cycle of **A**) an entity-specific (ES) plan and **B**) an ES plan rule.

context-sensitive means that the ES plan's state is affected by changes in the application information, such as increasing the patient temperature. These state transitions are applied to the ES plan and its *knowledge action* component, which represents sets of modularized ECA rules.

When the ES plan is generated from the skeletal plan, the ES plan and its sets of rules go into *generated* state from the initial state as shown in Figure 3.4. In *generated* state, the ES plan is not yet a subject to execution, it should be firstly authorized to be then registered. So, it become subject to be executed. The ES plan is authorized by an domain expert, who is in charge of the ES plan to domain entities. Once it is authorized, the ES plan and its sets of rules go into the *registered* state. In the registered state, all rules of the ES plan are installed in the system. In this state, no rule has fired yet.

On the first occurrence of an event of interest to one or more of the ES plan's rules, the ES plan goes into the *active* state, and one or more rules are fired and go into the *executed* state. The *active* state includes two sub-states, *waiting* and *executing*, as shown in Figure 3.4.A. In the *waiting* state, all ES plan's rules are waiting for events that are of interest to them. In the *executing* state, at least one rule is being executed. Once the rule execution completes, the ES plan returns to

the waiting state. Between the *waiting* and *executing* states of the ES plan, the rules are considered to be executed, as shown in Figure 3.4.B . The *executed* state is a state for the rules. On this state, a rule is being executed and after the execution the rule was waiting to the next event occurrence of interest. The ES plan might be transited from *active* state to *inactive*, *terminated*, or *completed* states, as shown in Figure 3.4.A.

The *inactive* state means that all the ES plan rules become disabled. The ES plan might be transited from *inactive* state to *active* state. That means enabling the rules of the ES plan. The *terminated* state means that all the ES plan rules removed from the system, but are not removed from the ES plan itself. The *completed* state of a rule means that the execution of the rule successfully done and the rule will not be subject to any further execution. When all the enabled rules in the ES plan are completed that means the ES plan goes into the *completed* state. The *completed* state of the ES plan could be determined by a domain user, who is in charge of the ES plan. After the ES plan had become in the *completed* state, all the ES plan rules are removed from the system. It could be decided to re-register the ES plan again, after it had been terminated or completed.

3.3 The SIM Framework for Managing the Complex Information

This section presents the SIM framework for managing the complex information. The SIM framework is a generalized and enhanced version of the SpEM framework developed in an early stage of previous research by Dube (2004). The complex information goes through three phases: *Specification* to specify the skeletal plans; *Instantiation* to instantiate the entity-specific plans; and *Maintenance* to maintain the entity-specific plan during its life-cycle. The SIM framework consists of three planes, specification, instantiation, and maintenance planes with the human-

computer interaction support as a base, as illustrated in Figure 3.1. The rational and functionality of each plane are discussed in the following subsections.

3.3.1 The Specification Plane

The specification plane provides support for capturing the domain knowledge, from which the complex information at the level of skeletal plan is defined. This plane contains two main functions, *Capturing* and *Formalisation*.

3.3.1.1 Capturing

The capturing process aims at gathering domain knowledge as a pre-process to specify the complex information at a generic level as skeletal plans. This process involves the formalization of human knowledge regarding a certain activity to build a system that can guide the end user through performing a specific activity. The domain knowledge is provided in non-computer-interpretable form. That is a major obstacle to exchange domain knowledge among organisations and/or individuals. A standard computer-interpretable form is required to overcome this obstacle. There is a need to computer-based tools assisting in capturing domain knowledge. These tools are intermediator between the real-world and the computer-world.

3.3.1.2 Formalisation

In order to effectively support the SIM approach to model the complex information as skeletal plans, the specification plane must provide a computer-interpretable model for expressing the skeletal plans. This model supports automatic verification and validation of the complex information. Different methodologies, such as process based or event-driven based, could be used as a primitive represent ion of the *active* part of the complex information. Here, the primitive represention is following the Event-Condition-Action (ECA) rule paradigm.

The specification plane formally specifies the skeletal plan according to the conceptual model of the complex information discussed in the previous section. Therefore, the model of the skeletal plan should take into account the features of the skeletal plan, such as domain-specific and platform-independent.

3.3.2 The Instantiation Plane

This plane aims at refining the skeletal plan to suit an organization and generating entity-specific plans as the following.

3.3.2.1 Customisation

The customisation process of the complex information as skeletal plans is one of the most valued activities. It includes filtering, synthesising, and presenting the skeletal plans so that they are directly relevant to the client. Professional services firms generate an enormous amount of high-value domain knowledge, however, the final step of customizing this domain knowledge to meet the client specific situation arguably adds the greatest value to the process of incorporating domain knowledge into organization activities as complex information.

According to the SIM framework, the customisation is a process of adapting the skeletal plan to meet the customer‘s, organisation‘s, and/or environment specific needs. The customisation process provides support to the skeletal plan to be an adaptive template. Therefore, the SIM framework provides flexibility for the customisation process. Because the SIM framework provides support to the deviation from standards, based on which the skeletal plans are defined.

The customisation of skeletal plans provides the ability to balance the uniqueness of an organisation with domain knowledge that is in common with other organisations. This process required assisting tools that is able to formalize the needs of the

organization; to identify how far the skeletal plans is compatible with these needs; and to automatically adapt the skeletal plan to these needs.

3.3.2.2 Instantiation

The instantiation is the process of generating an entity-specific plan from the skeletal plan. The instantiation process should provide a model for the entity specific plan. This model should take into account the features distinguishing the entity specific plan from the skeletal, such as platform- and entity- specific. This process considers the information of specific entity and maps the four components of a skeletal plan into the corresponding component in the entity-specific plan at low level of details.

3.3.2.3 Realization

The realization is the process of activating the entity-specific plan in reality. After understanding and reviewing the entity-specific plan clearly and distinctly, the entity-specific plan is authorized to be in the condition of being in operation or service. This condition is achieved by installing or registering the knowledge action component in the system managing the domain entities. The realized entity-specific plans are maintained by the maintenance Plane.

3.3.3 The Maintenance Plane

The maintenance plane provides the means, which are needed to support life-cycle of entity-specific plans and keep the complex information in a functional state. That requires several functionality such as execution, manipulation, query, information mining, and distribution management.

3.3.3.1 Execution

The entity-specific plans are executed as soon as a change of interest happens. That requires a computer-based execution model, which depends on the *active* part representation model provided by the specification plane. The Event-Condition-Action (ECA) rule paradigm is adopted as a representation model for the primitives of the *active* part of the complex information. In the instantiation process of the entity-specific plan, the *knowledge action* component of the skeletal plan is mapped into platform-specific that is amenable to execution by using a specific execution environment, such as specific active DBMS.

The instantiation and realization processes are a pre-process for the execution. The instantiation process translate the primitive rules of the knowledge action component into database triggers. In the realization process, these triggers are registered in the system. That means the core part of the adopted execution model is managed by the active database. That poses major challenges for the active database, which provides a basic implementation of the ECA rule paradigm. That implementation has a number of limitations in its support of the ECA rule components (Ceri et al. 2000).

3.3.3.2 Manipulation

The manipulation is a process that provides the operations against the complex information, skeletal plans and entity-specific plans. The complex information is subject to same manipulation operation, as other kind of information. These operations are add, delete, and modify. However, these operations are performed at high-level of abstraction that deals with the complex information in the terms of its components. The life-cycle of the complex information and specially entity-specific plan are to be supported by several manipulation operations, such as activate,

deactivate, terminate and fire. For example, the *deactivate* operation transit the entity-specific plan from the *active* state to the *inactive* state.

3.3.3.3 Query

The query is a process that provides the ability to query against the complex information, skeletal plans and entity-specific plans. The complex information is subject to same queries, as other kind of information. Queries may be issued in order to obtain information about skeletal plan dealing with specific situations and/or entity-specific plan of specific entity.

These queries are issued at a high-level of abstraction and might combine entity-specific information as well, such as what are the plans of patients (the domain entity here is a patient), whose ages are greater than 50 years old, and whose blood pressure is high? What are the skeletal plans dealing with medulloblastoma, which is a tumor that arises from embryonic cells in the inner part of the brain, and its diagnosis depends on the type and location of the tumor? Querying the skeletal plans is important to support the functionality of the instantiation plane by directly access specific skeletal plan.

In addition to these kind of queries, the entity-specific plan is subject to replay queries for recovering the plan at a specific time point and review the plan evolution over a time period. The replay query support provides a motion picture that depicts the evolution of the complex information. The *evolution history* component represents several information scenes. The ability of replaying these information scenes enhances the reporting and decision-support capabilities in the organization. The replay queries provides support to find out information, such as the time at which the entity-specific plan became active, at which a rule is executed, why it is executed, what is the action made, and how many times a rule is executed. An ex-

ample of special query is: replay the patient plan of patient X over the time period from T1 to T2. These special queries (replay queries) require support for querying the *evolution history* component of the entity-specific plan.

3.3.3.4 Information Mining

The information mining targets the automatic discovery of information from an evolution history component of the entity-specific plan, which represents a real case study. This discovered information can be used to deploy new domain knowledge or as a feedback tool that helps in auditing, analysing and improving already enacted domain knowledge.

The information mining is helpful because it gathers information about what is actually happening according to an evolution history component of several entity-specific plans, and not according to what people think that is happening during the activity. The starting point of any information mining is an evolution history component and the ability of querying it. From an evolution history component, one can find out information about the time at which the entity-specific plan became active, at which a rule is executed, why it is executed, what is the action made, and how many times a rule is executed. Therefore, the information mining provides an objective picture that depicts possible situations in which actions are performed in a predefined order.

3.3.3.5 Sharing and Distribution

The sharing and distribution provide interoperability support for managing the complex information in highly heterogeneous, widely distributed, and fragmented context. This context brings together a geographically dispersed stockholders, who are participating in the management process of the complex information. The sharing and distribution support require an infrastructure components in a platform-

independent and technology-neutral way.

Sharing the complex information refers to exchange information among and deliver it to people in need. Regarding the skeletal plan, information sharing facilitates the domain knowledge dissemination. The major obstacle for sharing the information of entity-specific plan is that that sharing violate the privacy of the entity. For example, sharing a patient plan violates the patient privacy. Therefore, in the process of sharing the entity-specific plan, all the entity privacy must be preserved.

The distribution management provides support for distributed execution, manipulation, and query. The distributed manipulation and query should overcome the heterogeneity fragmentation of the information. The distributed execution requires distributed event detection, condition evaluation and action. The time difference between geographically dispersed organization and users should be taken into account in the executing time-based rules.

3.3.4 Human-Computer Interaction Support

A Human-Computer Interaction support is required to be provided for the three planes of the framework. It is difficult to the end users to understand and review the skeletal plans and the entity-specific plans at the low level. The nature of the complex information as a huge amount of advanced information should be considered as an essential factor for the user interface in two directions. The first direction is to translate from a natural language, in context of domain knowledge, into a formal specification that the system can process further. The second direction is to translate the complex information from physical and low level representation into a human readable and high level representation model.

3.3.5 Complex Information Kernel

The core of the SIM framework is the complex information kernel, which are the the integrating factor and communication among the three planes. The complex information kernel provides a storage and retrieval support for the complex information. In this work, the DBMS and XML technologies are utilized as a base for the complex information kernel.

3.3.6 The SIM Framework Requirements

A high-level declarative language, which unifies the management of the three planes, is the main requirement for the SIM framework. The language should:

- provide a computer-interpretable model for the complex information as it is presented in the SIM approach. This computer-interpretable model supports at the same time the skeletal plan and the entity-specific plan and preserve their features;
- support the specification plan through incorporating domain knowledge as skeletal plans;
- provide support for customizing the skeletal plan and instantiating and realizing the entity-specific plan;
- provide a suitable mechanism for executing the entity-specific plan;
- provide both traditional and advanced manipulation operations and queries capabilities for the complex information; and
- be an interpretable language that is platform-independent

In order to support the human-computer interaction base, it is required to provide the ability of translating 1) the human language into the high-level declarative

language and 2) the computer-interpretable model complex information into a human readable model. It is also required to provide analytical language for mining the complex information.

Due to the distributed management nature, several modifications should be provided to the execution, manipulation and query of the complex information, such as 1) supporting the remote manipulation and query, and 2) providing a distributed detection for the situation of interest to the complex information.

3.4 Scope and Limitations

The scope of this thesis is the development of the SIM approach and the main management aspects provided in the SIM framework. These management aspects are:

- **in the specification plane**, the formalization of domain knowledge as skeletal plan. Capturing domain knowledge and its required management aspects are out of the scope of the thesis. As well As, the evolution history for the skeletal plan is not considered in the scope of the thesis.
- **in the instantiation plane**, the instantiation and realization of the entity-specific plan. The Customisation management aspects are out of the scope of the thesis.
- **in the maintenance plane**, the execution of the entity-specific plan, manipulation and query the complex information. The infrastructure for analysing and distributing the complex information is considered in this research. However, the management aspects for the information mining and distribution are out of the scope of the thesis. As well as, the human-computer interaction support are also out of the scope.

The focus of the thesis is to develop a unified language that facilitates the formalization, instantiation, execution, manipulation, and query the complex information with providing an infrastructure for the other management aspects stated under the SIM framework. The extensions required to the technologies adopted as a method for developing the framework and language are within the scope of the thesis.

The unified language supporting the SIM approach and framework is restricted to be applied to reactive applications that monitor events of interest to domain users, and respond to changes in situations by issuing alerts, reminders, requests, and/or observations to the domain user. The language provides the necessary information needed to make informed decisions. SIM combined with the unified language is evaluated using a proof-of-concept system utilized to manage a clinical case study of an health-care reactive application.

3.5 The Role of Temporal Active XML Database in Supporting SIM

Realizing SIM as a unified approach and framework for managing the complex information requires enabling technologies that: 1) can be seamlessly integrated and easily incorporated; 2) support the monitoring process; 3) support temporal data management; 4) provide interpretable support; 5) provide an integration support with the systems managing an application domain information.

A Temporal Active XML database is an XML database that includes active rules, in the form of ECA rules, and built-in time aspects for both XML data and ECA rules, e.g. temporal ECA rule model, a temporal data model and a temporal version of a query language. The XML database provides storage and retrieval support for XML data. The modern Database Management Systems (DBMSs), such as Oracle,

DB2 and SQL Server, provide a basic and primitive support for temporal data management, ECA rule paradigm and XML technologies. The modern DBMSs provide ECA rule-processing capabilities that supports monitoring and alerting processes. The provided ECA rule-processing capabilities are needed to be extended in order to deal with real-world situations. The temporal support provided by the modern DBMSs is very basic, because the modern DBMSs did not provide a temporal data model for storing and retrieving the history. The modern DBMSs are widely used in managing information of several application domains. Consequently, the Temporal Active XML database, as enabling technologies for information management, satisfies the five conditions stated in the previous paragraph.

The Temporal Active XML database is utilized here to support the SIM approach of modelling the complex information. It is adopted to play a crucial role in providing the base support for the three planes of the SIM framework and its base of Human-Computer interaction. The benefits of this method include: 1) the flexibility of managing the complex information as one unit (document), and the easy integration of the complex information management system into other systems. This method facilitates the development of the proposed complex information model and a management language and a decentralized system for the complex information.

Based on this method a language, called AIM, is developed as a high-level language that required to facilitate the management aspects of the SIM framework. Chapter 4 discusses the details of the AIM language, which:

- is an XML-based language and enjoys the general benefits of XML, such as parser reuse, incorporation into Web services, query generation;
- has an ECA- and XML- based specification component language, called AIM-SL, which formalizes the complex information into interpretable format;

- has a high level XQuery-based component language, AIMQL, that provides support to manipulate and query the complex information;
- provides a physical model for the complex information based on the temporal active XML;
- provide an execution mechanism based on translating ECA rules represented in the skeletal plan into triggers stored in the entity-specific plan.

Intermediate models for extending the modern DBMSs to support the temporal active XML method in a real-world context are developed. These models mainly extend the DBMSs to support temporal ECA rules and temporal XML model. These models are utilized by a proof-of-concept system, called AIMS, to implement the AIM language.

3.6 Chapter Summary

This chapter has presented the SIM approach and framework for managing the complex information. The SIM approach focuses on modelling the complex information at different abstraction levels (generic level and entity-specific level). The skeletal plan refers to the complex information at the generic level. The entity-specific plan refers to the complex information at the entity-specific level. The skeletal plan is to be instantiated to suite a particular entity and an entity-specific plan is generated. The SIM approach provides a conceptual model for the complex information and differentiates between the skeletal plan and the entity-specific plan.

The SIM framework provides comprehensive management aspects for managing the complex information. In the SIM framework, the complex information goes through three phases, specifying the skeletal plans, instantiating entity-specific plans, and then maintaining these entity-specific plans during their lifespan. Con-

sequently, these management aspects are classified into three planes, *specification*, *instantiation*, and *maintenance*. The specification plane includes the *capturing* and *formalizing* aspects. The instantiation plane includes the *customisation*, *instantiation*, and *realization* aspects. The maintenance planes includes the *execution*, *manipulation*, *query*, *information mining*, and *sharing and distribution* aspects. The base of the three planes is a human-computer interaction support.

The management aspects *capturing*, *customisation*, *information mining*, *sharing and distribution* and the *human-computer interaction* support are outside the thesis scope. However, providing an infrastructure for these management aspects is within the scope of the thesis.

The work done in this thesis is restricted to complex information within reactive applications with a support for the decision making process by providing the necessary information required for such a process. This information are provided through alerts and/or reminders. The thesis is unique in utilizing the temporal active XML database, which is a database providing support for ECA rule and XML with temporal data management, as a method for implementing the SIM approach and framework.

4

AIM: An Advanced Information Management Language for the Complex Information

This chapter presents a langauge, called AIM, for supporting the main management aspects of the SIM approach and framework. AIM is a complex information specification and query language. AIM is an acronym for **A**dvanced **I**nformation **M**anagement. The purpose of developing AIM is to facilitate the SIM approach and framework by supporting the complex information specification, instantiation, manipulation, and query.

The AIM language consists of three main components; specification component, instantiation model and query component. The specification component provides a computer-interpretable model and language, called AIMSL, for specifying the skeletal plan. The AIM language supports the instantiation and realization processes of the SIM framework by providing a computer-interpretable model, called ESP-Doc, for the entity-specific plan. This model combined with a database triggering

mechanism supports the entity-specific plan execution. The query component provides the AIM Query Language (AIMQL) that is used to manipulate and query the complex information (skeletal plan and entity-specific plan). The AIM language is a high-level, declarative and XML-based language. The AIM grammar syntax is defined using the XML Schema (van der Vlist 2002), and the AIM specifications are represented as XML document.

The chapter is organized as follows: Section 4.1 presents the AIM specification component that provides a specification language, called AIMSL; Section 4.2 discusses the AIM ESPDoc model for the entity-specific plan, and the AIM execution mechanism; Section 4.3 presents the AIM query component that supports manipulation and query processes; and Section 4.4 summarises the chapter.

4.1 The AIM Specification Component

The AIM specification component provides a specification language (AIMSL) for formalizing the domain knowledge as skeletal plans defined by the SIM approach in Chapter 3. AIMSL is an acronym for AIM Specification Language. AIMSL is the second stage of an ongoing work that starts with PLAN language (Wu and Dube 2001), which is based on the event-condition-action (ECA) rule paradigm.

The PLAN specification is represented in a plain text. Querying and manipulating a text file is limited to specific functions, such as find and replace functions. It is very important to provide query and manipulation support for the domain knowledge specification. In order to provide such support, TOPS (Dube 2004) maps the PLAN specification plain text into a database schema to be stored and managed using the DBMS. However, mapping the PLAN specification into relational database schema decomposes the specification into several tables. Therefore, it is not easy to deal with the specification as one document, as it is in the real life. Moreover, it

is not easy to exchange the specification between heterogeneous systems. As well as, representing the specification at different levels of abstractions is not supported in the PLAN language.

AIMSL enhanced the PLAN language mainly by enriching the ECA rule component, and extended the PLAN language to be an XML-based language. AIMSL overcomes the plain text limitations of the PLAN language by utilizing the XML Schema and XML langauge to represent the AIMSL grammar and specification, respectively as discussed in the following sub-sections.

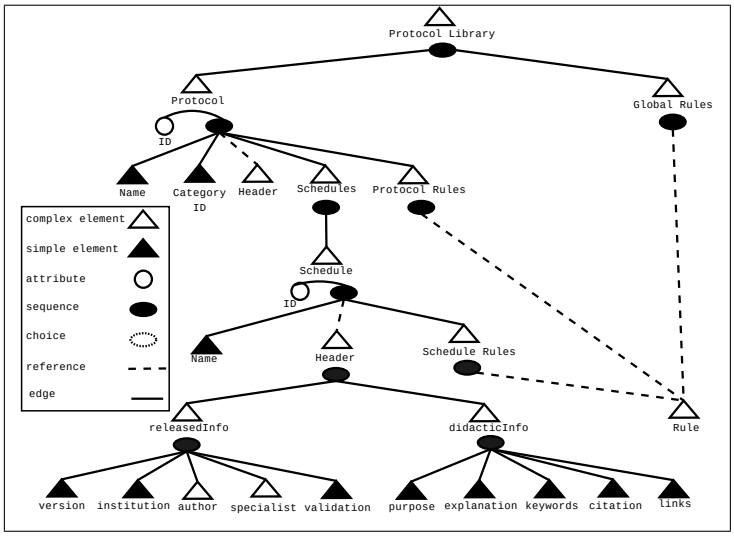

Figure 4.1: The AIMSL model based on XML Schema for the skeletal plan defined by the SIM Approach.

4.1.1 The AIMSL Model

AIMSL utilized the conceptual model of complex information provided by the SIM approach to provide a computer-interpretable model for the skeletal plan. Figure 4.1 depicts the AIMSL model that preserves the four components of the conceptual model of complex information, *knowledge action*, *domain information*, *evolution history*, and *descriptive information*. The *knowledge action* is implemented through

the *schedule* element, which is a modularized set of rules. The model of AIMSL follows the event-condition-action (ECA) rule paradigm. Figure 4.2 illustrates the AIMSL ECA rule paradigm.

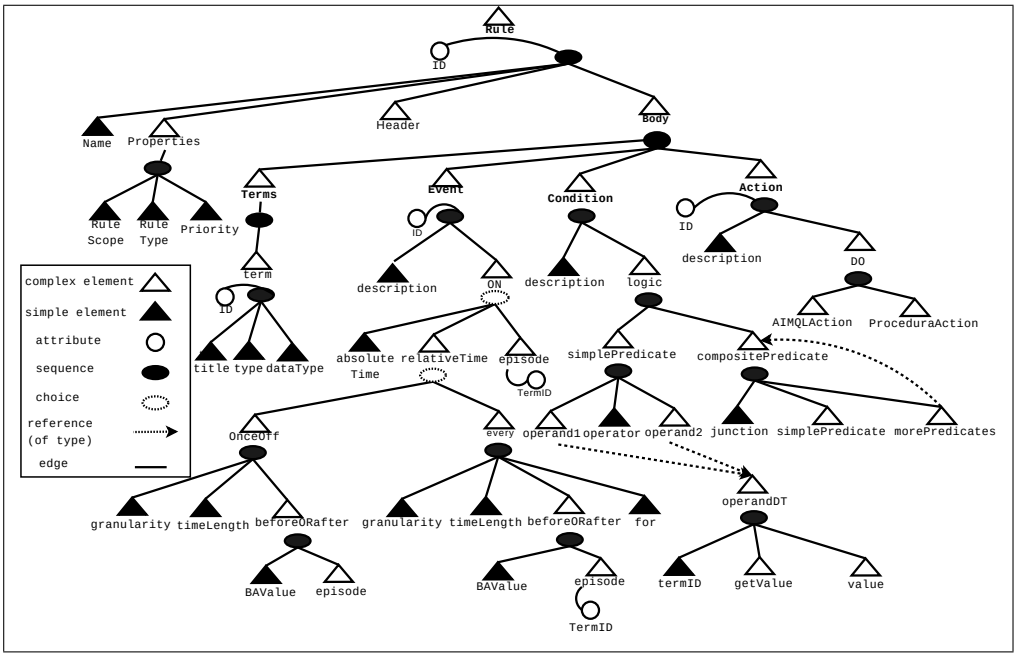

Figure 4.2: The XML Schema of AIMSL ECA rule paradigm.

AIMSL model expresses the best practice as modularized sets of rules, which are classified according to functional objectives and scopes. The *domain information*, which describes monitored information in order to detect a specific situation, is implemented through the *Terms* element. The *descriptive information* is implemented through the *Header* element. For simplicity, the *evolution history* component is not considered. For short, the AIMSL model refers to the skeletal plan as protocol.

4.1.1.1 Overview

As illustrated in Figure 4.1, the AIMSL model formalizes the domain knowledge as a protocol library, which consists of protocol (skeletal plan) specifications as well as specifications of global rules whose scope is the entire domain of discourse. As

shown in Figure 4.1, the individual protocols made up of schedules and a set of protocols rules that are not associated with any schedule. Each schedule is a set of rules that differs from an ordinary rule set in that it has an entry criteria and the fact that all rules in it are bound together by a common functional objective.

Each rule in a specification is deemed to be an ECA rule, which is defined over some relevant domain information attributes. It should also be noted that *protocol, schedule* and *rule* elements in the schema model has a set of attributes and that each element in the schema is made up of a sequence of a combination of attributes and other elements. Thus, the schema model allows ECA rules to be specified as either a memes of a set or a part of a *protocol* or a *schedule* elements. It should be pointed out here that the *protocol* and the *schedule* elements are manageable as single units although they are effectively sets of rules.

The AIMSL schema is modularized to provide flexibility in modifying or enriching the AIMSL language to suit several application domains. For example, the *condition* element shown in Figure 4.2 could be replaced with another element in order to suit a specific application domain.

4.1.1.2 Knowledge Action and Domain Information

The *knowledge action* and *domain information*, which are defined in Chapter 3, together describe actions, which should be taken, in a specific situation. The *domain information* is modelled using the *Terms* element that contains terms used in an application domain to be used in AIMSL specification and maps these terms into a specific database schema, which is used to manage the application information, consider as example the database schema of the patient record. The *knowledge action* is modelled as *Schedule* elements that contains rules. These rules contains three major elements, event,condition and action. The *Event* element defines the

context in which the AIMSL rule is relevant. The *condition* element analyses the application information and decides whether a specific action can be taken or not. The *action* element defines what is the appropriate action is, such as sending an email, changing in the *knowledge action* information.

4.1.1.3 Descriptive Information and Evolution History

The *descriptive information* is represented using the *Header* element that provides the necessary documentation for each skeletal plan and its sub-elements. The *descriptive information* facilitate the sharing of the skeletal plans. The *Header* element is a collection of pieces of release and didactic information. The release part provides information related to a specific specification version. The didactic part provides literature related to the domain knowledge; cites references to the source of the knowledge that is encapsulated in the AIMSL specification; and provides explanation. The *evolution history* component could be supported by providing extending the elements representing the *knowledge action* and *domain information* to be temporal elements that are able to log its changes over times.

4.1.2 The AIMSL language

The syntactic structure of the AIMSL language is specified using an XML Schema that follows the AIMSL model depicted in Figure 4.1. Instead of the Backus-Naur Form (BNF), the XML Schema (van der Vlist 2002; Fallside and Priscilla 2004) is used to formalize the syntactic structure of the AIMSL language. The XML Schema expresses the grammar of the AIMSL language at the element and attribute level, not at the character sequence level.

The AIMSL language consists of five main elements: *protocolLibrary*, *protocol*, *header*, *schedule*, and *rule*. The AIMSL language describes primitive reactive de-

cision logic of the domain knowledge for a specific situation as rules. The *rule* element should be expressive in order to express real-world situations and actions. The following sub-sections presents the AIMSL main elements.

4.1.2.1 Protocol Library

The *protocolLibrary* element is a library of computer-interpretable domain knowledge, which formalized as skeletal plans, to which the *protocol* element is a computer-interpretable model. The *protocolLibrary* element consists of global rules and protocols. Figure 4.3.A illustrates the AIMSL XML Schema for the *protocolLibrary* element.

The *protocolLibrary* element has a complex type composed of a sequence of two elements *protocols* and *globalRules*. The *protocols* element must appear exactly one time in the sequence. The *globalRules* element is optional, it may occur zero times. The *protocols* element has a complex type composed of a sequence of one *protocol* element or more. The *protocol* element should appear at least one time. That means the *protocolLibrary* element must contain at least one protocol (skeletal plan). The *globalRules* element has a complex type composed of a sequence of one *rule* element or more. The *rule* element should appear at least one time. Figure 4.3.B illustrates an example for a *protocolLibrary*, which consists of 7 protocols and 5 global rules.

4.1.2.2 Protocol

The *protocol* element is a computer-interpretable model of the skeletal plan, which is a logical framework and adaptive template for the domain knowledge utilized in a specific activity. Figure 4.4.A illustrates the AIMSL XML Schema of the *protocol* element. The AIMSL language is used to formalized the domain knowledge as several skeletal plans (protocol).

The *protocol* element has a complex type composed of a sequence of elements

```
<xsd:element name="protocolLibrary">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="protocols">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="pxsd:protocol" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="globalRules" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="rxsd:rule" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

A

```
-<protocolLibrary>
  -<protocols>
    +<protocol id="proID1">
    +<protocol id="proID2">
    +<protocol id="proID3">
    +<protocol id="proID4">
    +<protocol id="proID5">
    +<protocol id="proID6">
    +<protocol id="proID7">
  </protocols>
  -<globalRules>
    +<rule id="grul1">
    +<rule id="grul2">
    +<rule id="grul3">
    +<rule id="grul4">
    +<rule id="grul5">
  </globalRules>
</protocolLibrary>
```

B

Figure 4.3: A: the XML Schema definition for the protocol library. B: a protocol library example.

(*name*, *categoryID*, *header*, *schedules*, and *protocolRules*) and *id*. The value of the *name* element denotes the protocol name. The value of the *categoryID* element denotes the category of the protocol. The protocols and the domain entities are classified categories, based on which a specific protocol is used with an entity of the same category. The *header* element is explained in Sub-section 4.1.2.3.

The *schedules* element has a complex type composed of a sequence of one *schedule* element or more. The *schedule* element should appear at least one time. The *protocolRules* element is an optional element that has a complex type composed of a sequence of one *rule* element or more. The *rule* element should appear at least one time. Figure 4.4.B illustrates an example for a *protocol*, which consists of 7 schedules and 5 protocol rules.

4.1.2.3 Header

The *header* element provides descriptive information regarding an element, to which the *header* element is attached. The *header* element is subject to changes over time. As shown in Figure 4.5.A, the *header* element has a complex type composed of a sequence of elements (*releaseInfo* and *didacticInfo*). The *releaseInfo* element has

```
<xsd:element name="protocol">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="categoryID" type="xsd:token"/>
      <xsd:element ref="hxsd:header"/>
      <xsd:element name="schedules">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="sxsd:schedule" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="protocolRules" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="rxsd:rule" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
  </xsd:complexType>
</xsd:element>
```

A

```
-<protocol id="proID1">
  <name>protocol1</name>
  <categoryID>CID316</categoryID>
  +<header>
  -<Schedules>
    +<schedule id="schID1">
    +<schedule id="schID2">
    +<schedule id="schID3">
    +<schedule id="schID4">
    +<schedule id="schID5">
    +<schedule id="schID6">
    +<schedule id="schID7">
  </Schedules>
  -<protocolRules>
    +<rule id="prulID1">
    +<rule id="prulID2">
    +<rule id="prulID3">
    +<rule id="prulID4">
    +<rule id="prulID5">
  </protocolRules>
</protocol>
```

B

Figure 4.4: A: the XML Schema definition for the protocol. B: a protocol example.

a complex type composed of a sequence of elements (*version*, *institution*, *author*, *specialist*, *validation*). The *didacticInfo* element has a complex type composed of a sequence of elements (*purpose*, *explanation*, *keywords*, *citation*, *links*). According to the needs of a specific application domain, more pieces of information could be added under the both elements; (*releaseInfo* and *didacticInfo*); to make the header is more useful. The *header* element is a sub-element for the *protocol*, *schedule*, and *rule* elements. The header element is optional sub-element. That is to give flexibility in adding the descriptive information at any time of the specification and formalization process. Figure 4.5.B illustrates an example for a header.

Figure 4.6.A shows the definition of the *personDT* datatype. The *personDT* datatype is a complex type composed of elements (*name*, *email* and *contactNumber*). The *validationDT* datatype is a simple type that restricts the token datatype to the values (*production*, *research*, *test* and *expired*). The *production* value means that it is approved for applying to domain entities. The *research* value means that it is approved for research only. The *test* value means approved for test. The *expired* value means that it is no longer in use. Figure 4.6.B illustrates an example for a

```
<xsd:element name="header">
  < xsd:complexType>
    < xsd:sequence>
      < xsd:element name="releaseInfo">
        < xsd:complexType>
          < xsd:sequence>
            < xsd:element name="version" type="xsd:integer"/>
            < xsd:element name="institution" type="xsd:string"/>
            < xsd:element name="author" type="personDT" minOccurs="0"/>
            < xsd:element name="specialist" type="personDT" minOccurs="0"/>
            < xsd:element name="validation" type="validationDT"/>
          < /xsd:sequence>
        < /xsd:complexType>
      < /xsd:element>
      < xsd:element name="didacticInfo">
        < xsd:complexType>
          < xsd:sequence>
            < xsd:element name="purpose" type="xsd:string"/>
            < xsd:element name="explanation" type="xsd:string"/>
            < xsd:element name="keyWords" type="xsd:string"/>
            < xsd:element name="citation" type="xsd:string"/>
            < xsd:element name="links" type="xsd:string"/>
          < /xsd:sequence>
        < /xsd:complexType>
      < /xsd:element>
    < /xsd:sequence>
  < /xsd:complexType>
< /xsd:element>
```

A

```
<header>
  <releaseInfo>
    <version>1</version>
    <institution>James Hospital</institution>
    +<author>
    +<specialist>
    <validation>test<validation>
  </releaseInfo>
  <didacticInfo>
    +<purpose>
    +<explanation/>
    +<keyWords>
    +<citation>
    +<links/>
  </didacticInfo>
</header>
```

B

Figure 4.5: A: the XML Schema definition for the header. B: an header example

specialist of type personDT.

```
<xsd:complexType name="personDT">
  <xsd:sequence>
    <xsd:element name="name">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="firstname" type="xsd:string"/>
          <xsd:element name="surname" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="email" type="xsd:string"/>
    <xsd:element name="contactNumber" type="xsd:token"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="validationDT">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="production"/>
    <xsd:enumeration value="research"/>
    <xsd:enumeration value="test"/>
    <xsd:enumeration value="expired"/>
    </xsd:restriction>
</xsd:simpleType>
```

A

```
<specialist">
  <name>
    <firstname>Essam</firstname>
    <surname>Mansour<surname>
  </name>
  <email>emansour@dit.ie<email>
  <contactNumber>4024701<contactNumber>
<specialist">
```

B

Figure 4.6: A: the XML Schema definition for the person and validation datatype. B: an example for a specialist of type personDT

4.1.2.4 Schedule

AIMSL language formalizes the reactive behaviour extracted from the domain knowledge as modularized sets of the AIMSL ECA rules. Each set is represented as a *schedule* element, which carries out a specific function and may be used alone or combined with other sets. Modularizing the AIMSL ECA rules facilitates the manipulation of these rules.

Figure 4.7.A shows the XML Schema of the *schedule* element that has a complex type composed of a sequence of elements (*name*, *header*, and *scheduleRules*) and *id*. The value of the *name* element denotes the schedule name. The *header* element is explained in subsection 4.1.2.3. The *scheduleRules* element is a mandatory element that has a complex type composed of a sequence of one *rule* element or more. The *rule* element should appear at least one time. That means the schedule should at least contain one rule. Figure 4.7.B shows an example for a *schedule* element.

```
< xsd:element name="schedule">
  < xsd:complexType>
    < xsd:sequence>
      < xsd:element name="name" type="xsd:string"/>
      < xsd:element ref="hxsd:header"/>
      < xsd:element name="scheduleRules" minOccurs="1">
        < xsd:complexType>
          < xsd:sequence>
            < xsd:element ref="rxsd:rule" maxOccurs="unbounded"/>
          < /xsd:sequence>
        < /xsd:complexType>
      < /xsd:element>
    < /xsd:sequence>
    < xsd:attribute name="id" type="xsd:ID"/>
  < /xsd:complexType>
< /xsd:element>
```

A

```
< schedule id="schID1">
  <name>schedule 1</name>
  +<header>
  -<scheduleRules>
    +<rule id="rulID1">
    +<rule id="rulID2">
    +<rule id="rulID3">
    +<rule id="rulID4">
    +<rule id="rulID5">
    +<rule id="rulID6">
    +<rule id="rulID7">
  </scheduleRules>
</schedule>
```

B

Figure 4.7: A: the XML Schema definition for the schedule. B: a schedule example.

4.1.2.5 Rule

The *rule* element represents the primitive reactive decision logic of the domain knowledge. The *rule* element is an ECA rule that should has event and action, and might have condition. AIMSL extends the ECA rule paradigm to support advanced features, such as temporal events and domain-specific events. The *rule* element is specified as a platform-independent statement that could be directly mapped into executable statements, such as SQL triggers (Widom and Ceri 1996) or XQuery triggers (Bonifati et al. 2002; Shao et al. 2004). Section 4.1.3 presents in details the the rule element as an AIMSL ECA rule paradigm.

Figure 4.8.A shows the XML Schema of the *rule* element that has a complex

type composed of a sequence of elements (*name*, *properties*, *header*, and *body*) and *id*. The value of the *name* element denotes the rule name. The *properties* element has a complex type composed of a sequence of elements (*ruleScope*, *ruleType*, and *priority*). The *priority* element determines the order in which the rule should be invoked.

The *ruleScope* element determines the rule scope, which is *global*, *protocol* or *schedule*. The *ruleScopeDT* is a simple datatype that restricts the token datatype to accept only 3 values (*global*, *protocol*, and *schedule*). Respectively, they refer to the three types of the rules, *global rule, protocol rule,* and *schedule rule*. A *global rule* is a rule carrying out actions irrespective of the protocol being followed for the patient. The *global rule* is associated with all protocols. The *global rule* is applied to all patients. A *protocol rule* is a rule carrying out actions irrespective of the schedule being followed for the patient. The *protocol rule* is associated with all schedules of the protocol. A *schedule rule* is a rule associated with the schedule only.

The *ruleType* element determines the rule type, which is *static* or *dynamic*. The *ruleTypeDT* is a simple datatype that restricts the token datatype to accept only 2 values (*static* and *dynamic*). They refer to the type of the rule. A *static rule* is a rule that has only time-based event and action. The *static rule* is useful in representing the actions associated with a time table. A *dynamic rule* is an ECA rule that has event, condition and action.

The *header* element is explained in subsection 4.1.2.3. The *body* element has a complex type composed of three elements (*terms*, *event*, *condition*, *action*). The *terms* element maps terms used in the *event*, *condition*, and *action* elements to the institutions database. The *event* element determines when the rule should be triggered. The *condition* element is an optional element that specifies the criteria,

which should be satisfied to perform the action. The *action* element determines the action that should be performed. These elements are discussed in more details in the next section. Figure 4.8.B shows an example for a *rule element*.

```
<xsd:element name="rule">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="properties">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="ruleScope" type="ruleScopeDT"/>
            <xsd:element name="ruleType" type="ruleTypeDT"/>
            <xsd:element name="priority" type="xsd:integer"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element ref="hxsd:header"/>
      <xsd:element name="body">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="tsxsd:terms"/>
            <xsd:element ref="exsd:event"/>
            <xsd:element minOccurs="0" ref="cxsd:condition"/>
            <xsd:element ref="axsd:action"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
  </xsd:complexType>
</xsd:element>
```

A

```
<rule id="rulID1">
  <name>rule 1</name>
  <properties>
    <ruleScope>schedule<ruleScope>
    <ruleType>dynamic<ruleType>
    <priority>1<priority>
  </properties>
  +<header>
  -<body>
    +<terms>
    +<event>
    +<condition>
    +<action>
  </body>
</rule>
```

B

Figure 4.8: A: the XML Schema definition for the rule. B: a rule example.

4.1.3 AIMSL ECA Rule Paradigm

Providing ECA rule paradigm at an application domain and end-user level assists the domain users or experts to specify their knowledge easily using their own terminologies. AIMSL rules provides a temporal support for the ECA rule paradigm at a domain and high level. The AIMSL ECA rule paradigm consists of rule *ID, name, properties, header,* and *body*, as shown in Figure 4.2 that illustrates the XML Schema of the AIMSL rule paradigm. The *properties* element specifies the rule scope, type, and priority. The *header* element indicates what the rule is about, and provides release information. This section discusses the *body* of the AIMSL rule. The *body* consists of elements (*terms*, *event*, *condition*, and *action*).

4.1.3.1 Terms

Domain specific terms are used in specifying the rule *event, condition*, and *action*. The *terms* element specifies general terms and maps them into particular data items according to the utilized database schema. Examples for domain specific terms are *patient admission*, *test result received*, and *test value*. The terms *patient admission* and *test result received* are of type *event*. The term *test value* is of type *element*. If the term is of type *event*, it will be mapped into database operation(s), such as insert, delete, update. If the term is of type element, it will be mapped into database attribute.

Figure 4.9.A illustrates the XML Schema of the *terms* element that has a complex type composed of a sequence of one *term* element or more. The *term* element has a complex type composed of a sequence of elements (*title*, *type* and *dataType*) and *id* attribute. The value of the *title* element denotes the term title, such as *patient admission*. The *type* element is of the "*termTypeDT*" datatype, which is a simple datatype that restricts the token datatype to accept only 2 values (*event* and *element*). The *dataType* element is an optional element of the "*dataTypeDT*" datatype, which is a simple datatype that restricts the token datatype to accept only the values (*char*,*integer*, *double*, *date*, *time*, and *time stamp*). Figure 4.9.B shows an example for two terms of type *event* and *element*, respectively.

```
<xsd:element name="terms">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="term" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="title" type="xsd:string" minOccurs="1"/>
            <xsd:element name="type" type="termTypeDT" minOccurs="1"/>
            <xsd:element name="dataType" type="dataTypeDT" minOccurs="0"/>
          </xsd:sequence>
          <xsd:attribute name="id" type="xsd:ID"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

A

```
<Terms >
  <term id="E2.1">
    <type>event</type>
    <title>ACR test Result Received</ title >
  </term>
  <term id="E2.2">
    <title>ACR test result value</ title >
    <type>element</type>
    <dataType>integer</dataType>
  </term>
</ Terms >
```

B

Figure 4.9: A: the XML Schema definition for the term. B: an example for two terms

```
<xsd:element name="event">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="on">
        <xsd:complexType>
          <xsd:choice>
            <xsd:element name="absoluteTime" type="xsd:dateTime"/>
            <xsd:element name="relativeTime" type="relativeTimeDT"/>
            <xsd:element name="episode" type="episodeDT"/>
          </xsd:choice >
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
  </xsd:complexType>
</xsd:element>
```

A

```
<event id="EID123">
  <on>
    <absoluteTime>2008-01-14T12:13:29</absoluteTime>
  </on>
</event>
```

B

Figure 4.10: A: the XML Schema definition for the event. B: an example for an event of the type absolute time.

4.1.3.2 Event

The event is something that happens at a given place and time or a phenomenon located at a single point in space-time. AIMSL supports three kinds of events, *domain-specific event* (*episode*), *relative time event*, and *absolute time event*. The *episode* is an event or a series of connected events happening in the domain and related to a domain entity, such as *patient admission* and *test result received*, which happen in the heath-care domain and related to a specific patient. in AIMSL, the *episode* is associated with a term of type *event*. The *relative time event* is a temporal event, whose time is related to an *episode* event. The *relative time event* is happening *once-off* or *repeatedly*. Examples of *once-off events* are such as **on** 2 days **after** *patient admission* and **on** 2 hours **before** the *surgery*. Examples of *repetitive events* are such as **every** 3 days after *patient admission* for 10 days, and **every** 10 hours **before** the *surgery*. Figure 4.10.A illustrates the XML Schema of the *event* element that has a complex type composed of one of the elements (*absoluteTime*, *relativeTime*, *episode*) and *id* attribute. Figure 4.10.B illustrates an example for an event of the type absolute time.

4.1.3.2.1 Absolute Time Event The *absoluteTime* element is of the *dateTime* simple type, whose value is a set of integer values representing a date and time, such

A

```
<xsd:complexType name="episodeDT">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="term" type="xsd:IDREF"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="relativeTimeDT">
  <xsd:choice>
    <xsd:element name="onceOff" type="baseRelativeTimeDT"/>
    <xsd:element name="every">
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:extension base="baseRelativeTimeDT">
            <xsd:sequence>
              <xsd:element name="for" minOccurs="0">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="granularity" type="granularityDT"/>
                    <xsd:element name="timeLength" type="xsd:integer"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:choice>
</xsd:complexType>
```

B

```
<event id="EID124">
  <on>
    <episode term= "T123">patient admission
    </episode>
  </on>
</event>

<event id="EID125">
  <on>
    <relativeTime>
      <onceOff>
        <granularity>day</granularity>
        <timeLength >3</timeLength>
        <beforeORafter>
          <BAValue>after</BAValue>
          <term id="T123">patient admission
          </term>
        </beforeORafter>
      </onceOff>
    </relativeTime>
  </on>
</event>
```

Figure 4.11: A: the XML Schema definition for the event types. B: examples for events of type episode and relative time once-off.

as yyyy-mm-dd**T**hh:mm:ss. The *absoluteTime* element is used to specify an event that is not related to specific domain event, such as making a specific test on May 15, 2008 at 15.30 hours. Figure 4.10.B illustrates an example for an absolute time event, whose value is 2008-01-14T12:13:29.

4.1.3.2.2 Episode Event The *episode* element is of *episodeDT* datatype that has a complex type composed of a simple content value, which denotes the episode name, and an *id* attribute, which refers to a specific *term* element of the type *event*, as depicted in Figure 4.11.A. Figure 4.11.B illustrates an example for an episode event.

4.1.3.2.3 Relative Time Event The *relativeTime* element is of *relativeTimeDT* datatype that has a complex type composed of a choice between two elements (*onceOff*, *every*). The *onceOff* element refers to non-repetitive temporal event, and is of *baserelativeTimeDT* datatype. As depicted in Figure 4.11.A, the *baserelativeTimeDT* datatype is a complex type composed of a sequence of elements (*granularity*, *timeLength*, *beforeORafter*). The *granularity* element is of the *granularityDT*

```
<xsd:complexType name="baseRelativeTimeDT">
  <xsd:sequence>
    <xsd:element name="granularity" type="granularityDT"/>
    <xsd:element name="timeLength" type="xsd:integer"/>
    <xsd:element name="beforeORafter">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="BAValue"/>
          <xsd:element name="episode">
            <xsd:complexType>
              <xsd:simpleContent>
                <xsd:extension base="xsd:string">
                <xsd:attribute name="id" type="xsd:IDREF"/>
                </xsd:extension>
              </xsd:simpleContent>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="granularityDT">
    <xsd:restriction base="xsd:token">
      <xsd:enumeration value="second"/>
      <xsd:enumeration value="minute"/>
      <xsd:enumeration value="hour"/>
      <xsd:enumeration value="day"/>
      <xsd:enumeration value="week"/>
      <xsd:enumeration value="month"/>
      <xsd:enumeration value="year"/>
    </xsd:restriction>
</xsd:simpleType>
```

A

```
<event id="EID126">
  <on>
    <relativeTime>
      <every>
        <granularity>hour</granularity>
        <timeLength >5</timeLength>
        <beforeORafter>
          <BAValue>after</BAValue>
          <term id="T124">surgery
          </term>
        </beforeORafter>
        <for>
          <granularity>day</granularity>
          <timeLength>3</timeLength>
        </for>
      </every>
    </relativeTime>
  </on>
</event>
```

B

Figure 4.12: A: the XML Schema definition for the event base Relative Time DT. B: an example for a repetitive time event.

datatype. As shown in Figure 4.12.A, the *granularityDT* datatype is a token type restricted to the values (*second*, *minute*, *hour*, *day*, *week*, *month*, *year*). The integer value of *timeLength* element refers the number of unites. The *beforeORafter* expresses the triggering time relative to the episode (term of type event).

Figure 4.11.B illustrates the AIMSL specification for the event *3 days after patient admission*. In this event, the value of *granularity* element is day, the value of the *timeLength* element is 3, and the *beforeORafter* contains the value after for the *BAValue* element and the *episode* element referes to the term *patient admission* of type event.

The *every* element refers to a *repetitive time event*, such as *every* 5 hours after the surgery for 3 days. The *every* element extends the *baserelativeTimeDT* datatype by adding new element named *for*. The *for* element is an optional element that has a

complex type composed of a sequence of two elements (*granularity* and *timeLength*). The *for* element determines the period of the repetition. If the *every* element does not have the *for* element: 1) the rule is expired by the end of the ESP plan, if the value of the *beforeORafter* element is *after*; or 2) the rule is expired by reaching the start time of the term, on which the rule is based, if the value of the *beforeORafter* element is *before*.

The *every* element consists of the elements (*granularity*, *timeLength*, *beforeORafter*, and *for*). The previous *repetitive time event* is shown in Figure 4.12.B, in which the assigned values to the elements (*granularity*, *timeLength*, *beforeORafter*, and *for*) are (hour, 5, surgery, (day and 3)), respectively.

```
<xsd:element name="condition">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="description" type="xsd:string" minOccurs="0"/>
      <xsd:element name="logic" minOccurs="1">
        <xsd:complexType >
          <xsd:sequence>
            <xsd:element name="simplePredicate" type="simplePredicateDT" minOccurs="1"/>
            <xsd:element name="compositePredicate" type="compositePredicateDT" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
  </xsd:complexType>
</xsd:element>
```

Figure 4.13: The XML Schema definition for the condition.

4.1.3.3 Condition

A condition is a logical expression meaningful to the domain users, and determines whether to perform an action or not. The historical and snapshot information are subject to be checked by a condition. As shown in Figure 4.13, the *condition* element has a complex type composed of a sequence of elements (*description* and *logic*) and *id* attribute. The *description* element is an optional element that explains the semantic of the condition. The *logic* element contains a sequence of elements (*simplePredicate* and *compositePredicate*), which are of datatypes *simplePredicat-*

eDT and *compositePredicateDT*, respectively, as illustrated in Figure 4.14. The *compositePredicate* element is an optional element.

```
<xsd:complexType name="simplePredicateDT">
  <xsd:sequence>
    <xsd:element name="operand1" type="operandDT"/>
    <xsd:element name="operator" type="logicalOperatorDT"/>
    <xsd:element name="Operand2" type="operandDT"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="compositePredicateDT">
  <xsd:sequence>
    <xsd:element name="junction" type="junctionDT" minOccurs="1"/>
    <xsd:element name="predicate" type="simplePredicateDT" minOccurs="1"/>
    <xsd:element name="morePredicate" type="compositePredicateDT" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

Figure 4.14: The XML Schema definition of the simple and composite predicate datatypes.

```
<xsd:complexType name="operandDT">
  <xsd:choice>
    <xsd:element name="termID" type="xsd:IDREF"/>
    <xsd:element name="getValue" >
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="of" type="xsd:IDREF"/>
          <xsd:element name="number" type="xsd:integer"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="value" >
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="amount" type="xsd:string"/>
          <xsd:element name="datatype" type="valueDT"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:choice>
</xsd:complexType>
```

Figure 4.15: The XML Schema definition of the operand1 and operand2 datatypes.

4.1.3.3.1 Simple Predicate *simplePredicate* expresses a condition of two operands that are connected using an operator (=, <>, >, >=, <, or <=). Examples to simple predicates are *test result* $Y < 25$, test result X >= test result Y, and 5th ACR test result > 55. The *operand1* and *operand2* elements might be:

- a refereance to a *term* element of type *element*, such as *ACR test result*;
- the *getValue* function that is applied to temporal data. In the case of receiving

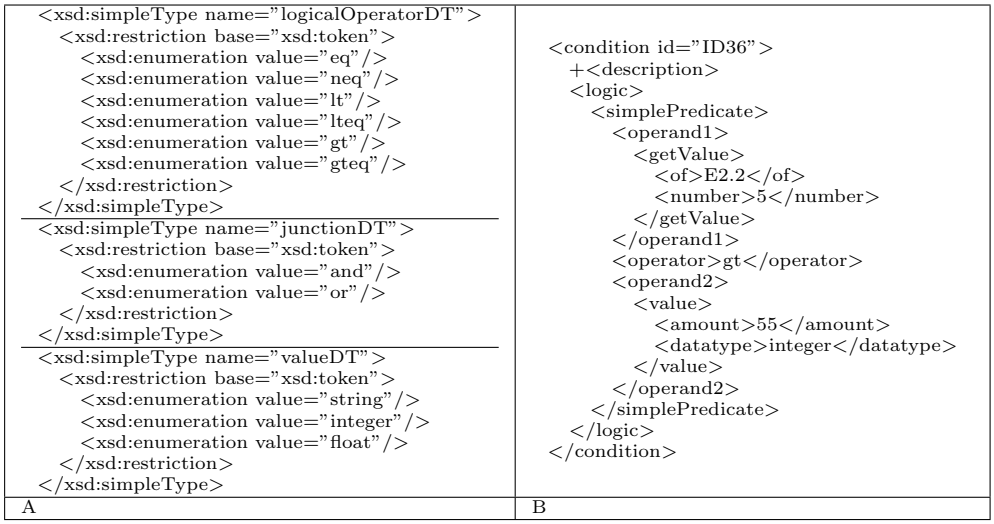

Figure 4.16: A: the XML Schema definition of the simple datatypes. B: an example for a simple condition.

several test results, it might be needed to evaluate the fifth test result. The *getValue* function returns the value number 5 of the *test result*; or

- a *value* of a specific datatype, such as 25 that is an integer value.

The *simplePredicateDT* datatype is a complex type composed of a sequence of elements (*operand1*, *operator*, *operand2*). Both *operand1* and *operand2* elements are of type *operandDT* that is a complex type composed of one element of (*termID*, *getValue*, or *value*) elements, as shown in Figure 4.15. The *termID* element is a references to a specific *term* element defined under the *terms* element. The *getValue* element has a complex type composed of a sequence of elements (*of* and *number*). The *of* element refers to a specific *term* element. The *number* element referes to a specific integer number. The *value* element has a complex type composed of two element (*amount* and *datatype*). The *datatype* element is of *valueDT* datatype that is a token type restricted to the values (*string*, *integer*, *float*), as shown in Figure 4.16.A.

For example, the value of *test result Y* should be an integer value. As shown in Figure 4.16.A, the logicalOperatorDT datatype is a simple type that restricts the token datatype to the values (*eq, neq, lt, lteq, gt,* and *gteq*). Respectively, they refer to equal, not equal, less than, less than or equal, greater than, and greater than or equal. Figure 4.16.B illustrates an example for a condition containing a simple predicate, which checks that the fifth value of a specific term is grater than 55.

```
<condition id="ID37">
  +<description>
  <logic>
    <simplePredicate>
      <operand1>
        <getValue>
          <of>TER123</of>
          <number>3</number>
        </getValue>
      </operand1>
      <operator>lt</operator>
      <operand2>
        <value>
          <amount>75</amount>
          <datatype>integer</datatype>
        </value>
      </operand2>
    </simplePredicate>
    <compositePredicate>
      <junction>and</junction>
      <predicate>
        <operand1>
          <getValue>
            <of>TER123</of>
            <number>3</number>
          </getValue>
        </operand1>
        <operator>gt</operator>
        <operand2>
          <value>
            <amount>55</amount>
            <datatype>integer</datatype>
          </value>
        </operand2>
      </predicate>
    </compositePredicate>
  </logic>
</condition>
```

Figure 4.17: An example for a composite condition.

4.1.3.3.2 Composite Predicate The *compositePredicateDT* datatype is a complex type composed of a sequence of elements (*junction*, *predicate*, *morePredicate*). The *junction* element is of type *junctionDT* that is a token type restricted to the values (*and* or *or*), as shown in Figure 4.16.A. The *predicate* element is of type *simplePredicateDT*. The *morePredicates* element is of type *compositePredicateDT*,

which provides support to express composite predicates. To expresses composite predicates, such as (P1 or P2) and (P3 and P4) or P5; where Pi is a simple predicate, the *logic* element contains:

- a *simplePredicate* element, which is P1; and
- a predicate element, which represents "or P2) and (P3 and P4) or P5" as *compositePredicate*.

Figure 4.17 illustrates an example for a composite condition, which checks that the third value of a term, whose ID is TER123, is less than 75 and grater than 55. The condition element contains a simple predicate, which checks that the term is less than 75, and a composite predicate, which connects the previous simple predicate using the *and* conjunction with the simple predicate, which checks that the the term is grater than 55.

```
<xsd:element name="action">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="description" type="xsd:string" minOccurs="0"/>
      <xsd:element name="do">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="AIMQLxsd:AIM-QLAction" minOccurs="0"/>
            <xsd:element name="proceduralAction" type="proceduralActionDT" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
  </xsd:complexType>
</xsd:element>
```

Figure 4.18: The XML Schema definition for the action.

4.1.3.4 Action

An action is an operation meaningful to domain users. The action element is a procedural action, such as sending email, or an AIMQL action for manipulating or querying the complex information. As shown in Figure 4.18, the action element has a complex type composed of a sequence of two elements (*description*, *do*) and *id*

```
<xsd:complexType name="proceduralActionDT">
  <xsd:sequence>
    <xsd:element name="SendSMS" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="mobileNo" type="xsd:integer"/>
          <xsd:element name="content" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="sendEMAIL" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="from" type="xsd:string"/>
          <xsd:element name="to" type="xsd:string"/>
          <xsd:element name="subject" type="xsd:string"/>
          <xsd:element name="content" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="invokeMethod" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
          <xsd:element name="parameters" type="xsd:string" minOccurs="0" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

Figure 4.19: The XML Schema definition for the procedural action.

A

```
<action id="AID5">
  <do>
    <proceduralAction>
      <sendEMAIL>
        <from>emansour@gmail.com</from>
        <to>emansour@gmail.com</to>
        <subject>ACR Test Order</subject>
        <content>make the ACR test to patient number PID1234</content>
      </sendEMAIL>
    </proceduralAction>
  </do> </action>
```

B

```
<action id="AID36">
  <do>
    <AIMQLAction>
      <add>
        <rule id="RUL1">
        <rule id="RUL2">
      </add>
    </AIMQLAction>
  </do>
</action>
```

Figure 4.20: A: an example for an action of a procedural type. B: an example for an action of a AIMQL type.

attribute. The *description* element is an optional element that explains the semantic of the action. The *do* element has a complex type composed of a sequence of two elements (*AIM-QLAction*, *proceduralAction*). the AIMQL actions are defined in the schema of the AIMQL language, as explained in Section 4.3.

Figure 4.19 illustrates the *proceduralActionDT* that is a complex type composed of elements (*sendEmail*, *sendSMS*, and *invokeMethod*). The *sendEmail* element has a complex type composed of two elements (*mobileNo* and *content*). They determine the content of the short message and the received number. The *sendEmail* element

has a complex type composed of a sequence of the elements (*from*, *to*, *subject*, *content*). Respectively, they specify the sender, the receiver, the subject of the email and the email content. The *invokeMethod* has a complex type composed of elements (*name* and *parameters*). Respectively, they specify the method name and the required parameters, if any.

Figure 4.20.A illustrates an example for an action of procedural type, where the action sends an email. Figure 4.20.B illustrates an example for an action of AIMQL type, where the action adds two rules two an AIMSL specification.

4.1.4 An Example

Figure 4.21 illustrates an example for two rules of the simplified version of the microalbuminuria screening protocol (MAP), which has a schedule containing two rules, MAP1 and MAP 2. MAP2 defines a set of clinical recommendation that should happen two hours after the result of the required test in MAS1 is received.

Rule MAP1:
ON day 2 after the patient admission,
DO order the test albumin creatine ratio (ACR)

Rule MAP2:
ON 2 hours after receiving the result of test ACR
IF the first ACR test result is greater than 25
DO order ACR test twice on
day number 6 after the patient admission and
day number 38 after the patient admission

Figure 4.21: Two rules of the microalbuminuria screening (MAS) protocol.

```
-<protocol id="ProID-MAP">
  <name>microalbuminuria screening protocol (MAP)</name>
  <categoryID>CID316</categoryID>
  +<header>
  -<Schedules>
    -<schedule id="SIDMAP">
      <name>Basic MAP</name>
      +<header>
      -<scheduleRules>
        +<rule id="MAP1">
        +<rule id="MAP2">
      </scheduleRules>
    </schedule>
  </Schedules>
</protocol>
```

Figure 4.22: the AIMSL specification for the simplified version of the microalbuminuria screening (MAS) protocol.

```
-<rule id="MAS2">
          <name>Rule 2 of basic MAP</name>
          +<properties>
          + <header>
          - <body>
            -<Terms >
              <term id="E2.1">
                <type>event</type>
                <title>ACR test Result Received</ title >
              </term>
              <term id="E2.2">
                <title>ACR test result value</ title >
                <type>element</type>
                <dataType>integer</dataType>
              </term>
            </ Terms >
            -<event id="E1R2">
              <on>
                <relativeTime>
                  <onceOff>
                    <granularity>hours</granularity>
                    <timeLength >2</timeLength>
                    <beforeORafter>
                      <BAValue>after</BAValue>
                      <term id="E2.1">ACR test Result Received</term>
                    </beforeORafter>
                  </onceOff>
                </relativeTime>
              </on>
            </exsd:event>
            -<condition id="ID36">
              +<description>
              <logic>
                <simplePredicate>
                  <operand1>
                    <getValue>
                      <of>E2.2</of>
                      <number>1</number>
                    </getValue>
                  </operand1>
                  <operator>gt</operator>
                  <operand2>
                    <value>
                      <amount>25</amount>
                      <datatype>integer</datatype>
                    </value>
                  </operand2>
                </simplePredicate>
              </logic>
            </condition>
            - <action id="AID36">
              - <do>
                -<AIMQLAction>
                  - <add>
                    +<rule id="MAS3">
                    +<rule id="MAS4">
                  </add>
                </AIMQLAction>
              </do>
            </action>
          </body>
</rule>
```

Figure 4.23: the AIMSL specification for the rule MAP2.

If the first result is greater than 25, the action of the rule MAP2 is executed, and adds two rules to the specification, MAP3 and MAP4. The both rules are similar to the rule MAP1, but they fire on day 6 and day 38 after the patient admission, respectively. Figure 4.22 illustrates the AIMSL specification for the simplified version of the microalbuminuria screening protocol (MAP), which contains one schedule containing two rules.

A focus is given to the specification of rule MAP2, whose specification is illustrated in Figure 4.23. Rule MAP2 has two *term*s elements. A *term* element of type

event represents the event *ACR test result received*. Another *term* element of type element represents the value *ACR test result value*. The event element of the *MAP2* Rule is a *once-off relativeTime* event based on the *ACR test result received term*, whose granularity is hour, and timeLength is 2. The *MAP2* rule is fired two hours after the *ACR test result received*.

The *logic* element of the *MAP2* condition is a *simplePredicate* that is *getValue* of the term *ACR test result value*, such that the value is the first value, and this value is grater than 25. The *action* element adds two rules; *MAP3* and *MAP4*.

4.1.5 Discussion

Section 4.1 has presented the AIM Specification language (AIMSL), and its model. AIMSL specification format is based on XML. AIMSL provides an advanced ECA rule paradigm. The following subsections discuss the merits of representing the AIMSL specification as an XML document, and the advanced features of the AIMSL *rule* element, which require an extension for the database triggering mechanism.

4.1.5.1 AIMSL Specification as an XML Document

AIMSL specification is stored as XML document to facilitate transport across dispart architecture. There is no need to use separation between the data items of the AIMSL specification, because of the tag-based representation of the XML document. AIMSL specifications are edited with many different kinds of editors, which are ranged from normal text or XML editors to an AIMSL editor.

A graphical AIMSL editor hides the code in the background and present the content to the user in a more user-friendly format. This is helpful for situations where people who are not fluent in AIMSL and XML code need to enter information in XML based documents. The AIMSL editor should take care of syntax details

by validating the AIMSL specification against the AIMSL model. The use of such editor is faster and more convenient.

4.1.5.2 Extension to the DBMS Triggering Mechanism

Translating the platform-independent rules (in the skeletal plan) into platform-specific rules (in the entity-specific plan) is a major challenge for providing an execution mechanism based on the active database, because the modern DBMSs provide a basic triggering mechanism, which has a number of limitations in its support of the ECA rule components (Ceri et al. 2000).

There is a need for intermediate models to extend the triggering mechanism of the modern DBMSs. AIMSL provides support for temporal events, which are absolute time events, and relative time events that is based on domain specific event, such as *patient admission*. In order to provide support for the time-based and domain-specific events, an extension to the event component of the DBMS trigger is needed to support the AIM execution mechanism. AIMSL provides support for temporal condition; that needs to extend the condition component of the DBMS trigger to evaluate temporal conditions. In order to support AIMSL action element, the action component of the DBMS trigger should be extended to allow detached actions that can be performed external to the DBMS and at some point long after the rule has been executed.

4.2 The AIM ESPDoc: an Instantiation and Execution Model for the Entity-Specific Plan

This section presents the AIM ESPDoc model, which provides a computer-interpretable model for the conceptual model of the entity-specific (ES) plan presented in Chapter 3, and the AIM execution mechanism. ESPDoc is an acronym for **E**ntity-**S**pecific

Plan **Doc**ument. AIM provides an execution mechanism based on active database to the ESPDoc model.

4.2.1 The AIM ESPDoc Model

the conceptual model of ES plan consists of two main parts an *active part* and the *passive part*. The *active part* represents the reactive behaviour derived from the skeletal plan, represented as an AIMSL protocol. The *passive part* represents the descriptive information, which represents the states of the ES plan and its evolution since it has been created.

The *passive part* is subject to actions that log the execution history of the ES plan. Therefore, the ES plan grows over time. The ES plan is subject to dynamically changes in order to suit the current conditions and constrains of interest to the domain user. The *active part* of the ES plan is represented as rules, which are coded as a trigger or several triggers, which are used to register the rule in the system. The *passive part* of the ES plan is modelled as time-varying information.

The AIM ESPDoc model provides support for the four component of the complex information, which are discussed in Chapter 3, at the level of the ES plan. The AIM ESPDoc is capable of storing the evolution history of the ES plan, as well as the descriptive information regarding the ES plan. The *knowledge action* and *domain information* components of a specific skeletal plan together are utilized to generate *rule* component of the ESPDoc model.

Figure 4.24 illustrates the XML Schema of the AIM ESPDoc model. As shown in Figure 4.24, the *knowledge action* and *domain information* components are represented as *rule* element. Each *rule* element contains a trigger or several triggers. The *evolution history* component is presented as *state* element. The *descriptive information* component is represented using the *header* element. The AIM ESPDoc

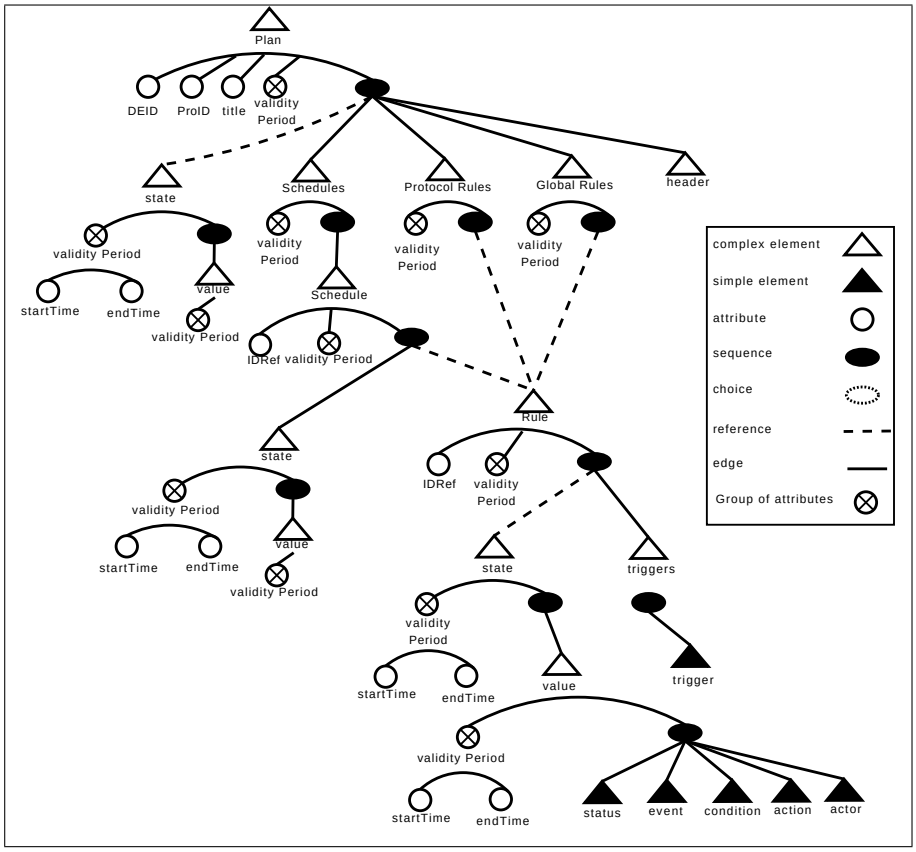

Figure 4.24: The AIM entity-specific plan model based on XML Schema.

is modelled as time-varying information. The model captures the valid times of the fact recorded under the ES plan. That is leading to temporal relations among the ES plan and its components. The validity period represents a time period, during which a component is existence as a part of the ES plan. A temporal XML support is needed to realize the AIM ESPDoc model.

4.2.2 Instantiation and Realization

The ES plan document is generated through two main steps instantiation and realization. The customization step is a pre-step, during which users apply some modifications, and/or attach more descriptive information in order to adapt the specified skeletal plan for use within a specific organization. In the instantiation

step, the creation of a ES plan document of a specific and appropriate skeletal plan is done. The instantiation process starts by accessing a specific protocol (skeletal plan specification), and construct the ES plan document according to its schedules, protocol rules, and the global rules. In the realization step, if the ES plan document is approved, its triggers are installed in the system.

Several varieties of languages could be utilized in creating the triggers, ranging from SQL language, to an active XML language or Web services combined with publish/subscribe technique. Chosen the language depends on the storage model and the nature of the application, whether it is decentralized or centralized. The AIM ESPDoc model is flexible to support varieties of languages.

4.2.3 Execution

The *evolution history* component, represented using the *state* element, is managed by the AIM ESPDoc model itself. This means the execution model of the AIM ESPDoc model should provide self-management support. The AIM execution method is based on the active and temporal mechanism.

4.2.3.1 Active Mechanism

The ECA rule paradigm as implemented in database systems is a promising technology for supporting the execution method of the ESPDoc model. The DBMSs provide support for the active mechanism using triggers. Once the ES plan rules are registered (installed) in a database system, the DBMS becomes in charge of executing the triggers representing the rules of an ES plan.

The semantic handling the temporal feature of ESPDoc is represented also as triggers. By this method a self-management for the ES plan model is provided. The Web service Notifications could be used to subscribe the monitored information,

which specified as general terms in the *Terms* element.

Utilizing the database triggering mechanism facilitates the integration of the AIM ESPDoc model into the system managing the domain information, such as the patient information system that manages the electronic healthcare record. The use of the Web Services provides support for distributed management for the ESPDoc model.

4.2.3.2 Temporal Mechanism

The ES plan and its component are joined with a validity period. The validity period refers to the period of existence, in which the component considered as part of the ES plan. It is assumed that the valid time, which is the time when the fact is true in the reality, equals the transaction time, which is the time when the fact is stored in the database. The validity period represents as a tuple,<start time, end time>. If the component has the validity period <5, NOW>, that means the component is currently part of the ES plan document since the time point 5.

Assume at time T2, the state of a component having a state S1 with validity period <T1, NOW>, is changed to S2. Then the new state is added to the component with validity period <T2, NOW> and the validity period of the old state will be <T1, T2>. That means at time T2, the state of the component is changed from S1 to S2. The validity period of a component is equal to <minimum (start time), maximum (end time)> of its sub-components. The details of the developed temporal XML data model utilized to support the AIM ESPDoc model is presented in Chapter 5.

4.2.4 An Example

An ES plan, in healthcare domain called patient plan, is generated based on the specified protocol shown in Figure 4.22. In the instantiation and realization process, the rule body (*terms, event, condition* and *action*) is used to generate a trigger, which could be encoded using SQL, SQL/XML, or XQuery triggering language. Choosing the triggering language depends on the type of the database used to store the domain information, whether it is a relational or XML database. Regarding the execution of this medical patient plan, it is assumed that:

- the medial patient plan is registered at time point 1; and
- the result of ACR test is received on day 3 and its value was 33, which is greater than 25.

According the the specification of rule MAP2, its action adds two new rules, MAP3 and MAP4, and then these changes are logged in the patient plan. Figure 4.25 illustrates part of the patient plan on day 4. This part has the history of the patient plan and its execution. The *state* element records the several states of a rule during the lif cycle of the plan. As shown in Figure 4.25, the rule *state* element might have several *value* elements. Each *value* element specifies a specific status to the rule, and specify the event firing the rule, the condition evaluated with the real values at the firing time, the action carried out, and/ or the actor participating.

For example, the first *value* element of rule *MAP1* contains only a *status* element, whose value is *generated*. The *generated* status is a system-defined status that happens at the generation time of an entity-specific plan. Therefore, there is not need to the other element, such as actor or event elements. Another example is the *value* element of rule *MAP3* that is registered by rule *MAP2*. The content of the *condition* element of rule MAP2 is "the condition (ACR test value is greater than

25) is true, because the ACR test value at the firing time was 33". The content of the *event* element of rule *MAP1* is " time-based rule fires when the plan is 2-hours old". These pieces of information, status, event, condition, action and actor, provide support in the reviewing process to know why and when the rule is fired and executed, how the rule is executed, who participates in moving the rule to such state.

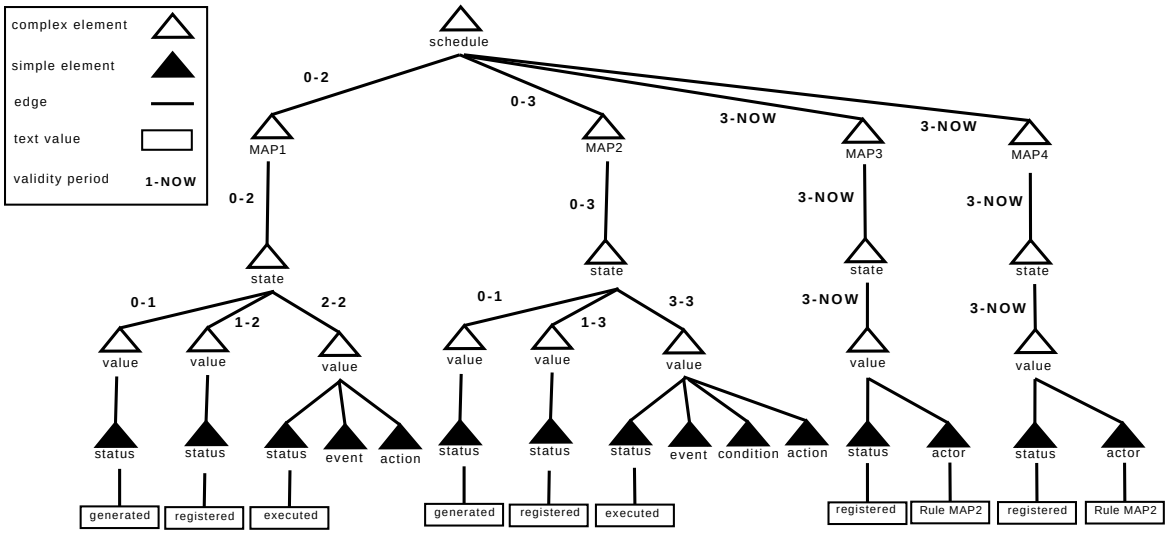

Figure 4.25: A part of the patient plan on day 4 after patient admission.

4.2.5 Discussion

This sub-section discusses the need to a replay support for the ES plans and a temporal XML support to realize the AIM ESPDoc model using the available DBMSs.

4.2.5.1 The need for a Replay Support

The entity-specific plans keep the execution history of applying specific domain knowledge. Such execution history represents several information scenes. The ability of replaying these information scenes enhances the reporting and decision-support capabilities in the organization. The replay support facilitates the infor-

mation analysis and mining to discover and understand the information trends.

The starting point for information analysis and mining is the evolution history component and the ability of replaying this history at a high and domain level. The replay support is to help to find out information, such as the time at which the entity-specific plan became active, at which a rule is executed, why it is executed, what is the action made, and how many times a rule is executed. Therefore, the replay support provides a motion picture that depicts the evolution of a specific task or activity. The AIM query component provides a replay support for the complex information, as discussed in the next section.

4.2.5.2 The need to a temporal XML Support

A temporal XML data model is required to support the AIM ESPDoc model. Several features should be addressed by the temporal XML data model, such as the temporal edges between XML elements, temporal elements, and the temporal constrains among the sub-elements and their parent element. In order to reuse the available XML DBMSs, the temporal XML data model should be compatible and consistent with the XML data model Therefore, all the XML tools could be used to deal with the ES plans specified using the ESPDoc model. The XQuery language could be used to query the ES plans, however the temporal relationships among the ES plan components should be considered. Moreover, the XML databases could be utilized to store and query the ESPDoc documents.

4.3 The AIM Query Component

There is a need to move the complexity of manipulating and querying the complex information (skeletal and entity-specific plans) from user/application code to a high level declarative language. AIMQL is a high level XQuery-based language

provides facilities to perform manipulation operations, and advanced queries, such as replaying dynamic execution scenarios of the complex information.

4.3.1 The Query and Manipulation Requirements of the Complex Information

The main functional requirements of AIMQL are to assist in: 1) Manipulating the AIMSL specification (skeletal plan) and ES plan. The changes are made to AIMSL specification might be required to be propagated to the corresponding ES plan; and 2) Retrieving this information. This includes the ability to replay the ES plan or a specific part of it within specific time period. There are general functional requirements that should be also provided to AIMQL. These requirements are:

- **Declarativity**, AIMQL should be declarative. It should be independent of any particular platform or query evaluation strategy;
- **Temporal Support**, it should be able to record the history of executing the ES plan reactive behaviour and to query it;
- **XQuery-based**, the AIMSL specification and ES plan are represented as XML document. Therefore, AIMQL should be based on XQuery; and
- **Convenient for humans to read and write**, this could be achieved using an XML-based graphical tool that assists in generating AIMQL query and browsing it.

XML is easy to be generated using tools and easy to be converted to human readable format using a stylesheet language, such as XSL. Using XML in representing AIMQL provides a compatibility with AIMSL, and assists in managing the complex information remotely, using Web services.

Several extensions to XQuery are required in order to achieve the AIMQL requirements as following:

- **Manipulation Operations:** AIMQL introduces seven manipulation operations (expressions). These expressions includes add, remove, modify, activate, deactivate, terminate and fire. The AIMQL manipulation operations are distinguished in the sense that they do not only potentially modify the AIMSL specification or ES plan, but also propagate the modification to the corresponding ES plan documents and modify the corresponding triggers created in the system. Furthermore, the manipulation expressions log the changes occurring to ES plan documents; and
- **Query Support:** AIMQL provides support to query AIMSL specification and ES plan document, as the domain information, plus special query capabilities, replay function and temporal query support for ES plan document. AIMQL introduces a new functionality called replay. AIMQL replay query is a query that plays over again the history of the complex information to show in details the actions that cause changes on the complex information and how it evolved over time.

		Skeletal Plan							
Category	**Function**	Cat	Pro	Sch	Rule	Trm	Eve	Con	Act
Manipulation	Add	A	A	A	A	A	A	A	A
	Remove	A	A	A	A	A	A	A	A
	Modify	A	A	A	A	A	A	A	A
	activate	X	X	X	X	X	X	X	X
	Deactivate	X	X	X	X	X	X	X	X
	Terminate	X	X	X	X	X	X	X	X
	Fire	X	X	X	X	X	X	X	X
Query	Normal	A	A	A	A	A	A	A	A
	Replay	X	X	X	X	X	X	X	X

Table 4.1: AIMQL function applicability for the skeletal plan

Tables 4.1 and 4.2 show the AIM manipulation and query support provided to the skeletal plan and entity-specific plan, respectively. The *A* value denotes that a feature is applied, and the *X* value denotes that a feature is not applied. The

		Entity-Specific Plan								
Category	**Function**	Ent	Pro	Plan	Sch	Rule	Trm	Eve	Con	Act
Manipulation	Add	X	X	X	A	A	A	A	A	A
	Remove	X	X	A	A	A	A	A	A	A
	Modify	X	X	X	A	A	A	A	A	A
	activate	X	X	A	A	A	X	X	X	X
	Deactivate	X	X	A	A	A	X	X	X	X
	Terminate	X	X	A	A	A	X	X	X	X
	Fire	X	X	X	X	A	X	X	X	X
Query	Normal	A	A	A	A	A	A	A	A	A
	Replay	X	X	A	A	A	X	X	X	X

Table 4.2: AIMQL functions applicability for the entity-specific plan.

columns (*Cat, Pro, Sch, Rule, Trm, Eve, Con, Act,* and *Ent*) shown in Tables 4.1 and 4.2 refer to (Category, Protocol, Schedule, Rule, Terms, Event, Condition, Action, domain entity) respectively. Each column represents a component of either the skeletal or entity-specific plan.

For the skeletal plan, the *add, remove* and *modify* operations are applied to all skeletal plan components. However, the *activate, deactivate, terminate* and *fire* operations are used to facilitate the execution of the entity-specific plan. Therefore, these operations are not used with the skeletal plan components, but used with the *plan, schedule* and *rule* components of the entity-specific plan. The *fire* operation is used only with the *rule* component. The entity-specific plan is generated for a specific domain entity from a specific protocol (skeletal plan). The domain entity and protocol of the entity-specific plan are not changeable. Therefore, the *add, remove* and *modify* operations are not applied to the domain entity nor the protocol components. Moreover, the *add* and *modify* operations are not applied to the *plan* component.

This research work focuses is on the execution history of the entity-specific plan. Consequentially, the AIMQL replay query is provided to the entity-specific plan, specially the components (*plan, schedule* and *rule*) that are called re-playable components. The other components of the entity-specific plan could be replayed as a part of the re-playable components.

4.3.2 The High-Level Manipulation Operations

The manipulation operations shown in Figure 4.26 are applied to the skeletal plan, entity-specific plan and its corresponding triggers created as an implementation for the execution process of this plan. The changes made to the skeletal plan or the entity-specific plan might need to be propagated to the corresponding plan or triggers, respectively. The manipulation operations could be issued in the action component associated with the AIMSL *rule* element.

```
<xsd:element name="manipulationOperation" >
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="add" type="modtxsd:addDT" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="remove" type="modtxsd:removeDT" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="modify" type="modtxsd:modifyDT" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="activate" type="modtxsd:activateDT" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="deactivate" type="modtxsd:deactivateDT" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="terminate" type="modtxsd:terminateDT" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="fire" type="modtxsd:fireDT" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4.26: The XML Schema definition of the AIMQL manipulation operations.

The supported manipulation operations are:

- **Add** a skeletal plan (protocol), entity-specific plan, or one of their components.
- **Remove** a protocol, entity-specific plan, or one of their components.
- **Modify** a protocol, entity-specific plan, or one of their components.
- **Activate** an entity-specific plan, schedule, or rule components.
- **Deactivate** an entity-specific plan, schedule, or rule components.
- **Terminate** an entity-specific plan, schedule, or rule components.
- **Fire** a rule component.

4.3.2.1 Add

The *add* operation is an manipulation operation that add copies of one or more protocol specification or ES plan components into a designated position with respect to a target component. Figure 4.27.A shows the XML schema of the *add* operation as follows:

```
<xsd:complexType name="addDT">
  <xsd:sequence>
    <xsd:element name="addedExpr" />
    <xsd:element name="as" />
    <xsd:element name="into">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="posBA"/>
          <xsd:element name="AddedTargetExpr"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="propagation" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

A

```
<add">
  <addedExpr>
    +<rule id="rul123">
  </addedExpr>
  <as>scheduleRule</as>
  <into>
    <AddedTargetExpr>
    protocol[id="pro123"]//schedule[id="sch123"]
    </AddedTargetExpr>
  </into>
</add">
```

B

Figure 4.27: A: the XML Schema definition of the add operation. B: an example for add operation.

- The *AddedExpr* represents one of the protocol (skeletal plan) or ES plan components.
- The *as* value could be one of this values (Category, Protocol, Schedule, Terms, Event, Condition, Action, or domain entity), or the values (schedule rule, protocol rule or global rule).
- the *AddedTargetExpr* represents a targeted component in a specific protocol or ES plan.
- If into is specified without *Before* or *After*, *AddedExpr* becomes children of the *AddedTargetExpr*. Else, *AddedExpr* becomes children of the parent of *AddedTargetExpr*.
- the *propagation* values are (Yes or No), and the default value is No. The *Added-*

Expr will not be propagated to the corresponding ES plans, if the value is No. If the value is Yes, the *AddedExpr* will be propagated to all the corresponding plans.

The semantics of an add expression are as follows:

- AddedExpr must be a valid AIMSL component for the protocol or ES plan; otherwise a static error is raised. The result of this step is either an error or a sequence of components to be added.
- AddedTargetExpr must refer to a valid AIMSL component; otherwise a static error is raised.
- The result of the add expression must be a valid AIMSL component for a protocol or ES plan; otherwise a dynamic error is raised.
- If the add expression is applied for a plan, the validity period associated with the *AddedTargetExpr* and its children should be changed to reflect the new changes that have been made by the add expression.

Figure 4.27.B shows an example for an *add* operation that adds a rule as a schedule rule under the schedule, whose id is *sch123* and its parent is a protocol, whose id is *pro123*. The added rule will not be propagated because the default value of the propagation is No.

4.3.2.2 Remove

A remove expression removes at least one of AIMSL components from a protocol or ES plan. Figure 4.28.A shows the syntax of a remove expression as follows:

- The RemovedTargetExpr refers to one of the protocol or ES plan components.

<xsd:complexType name="removeDT"> <xsd:sequence> <xsd:element name="RemoveedTargetExpr"/> <xsd:element name="propagation" minOccurs="0"/> </xsd:sequence> </xsd:complexType>	<remove"> <RemoveedTargetExpr> protocol[id="pro123"]//schedule[id="sch123"]//rule[id="rul123"] </RemoveedTargetExpr> <propagation>Yes</propagation> </remove">
A	B

Figure 4.28: A: the XML Schema definition of the remove operation. B: an example for a remove operation.

- the *propagation* values are (Yes or No), and the default value is No. The *RemovedTargetExpr* will not be propagated to the corresponding ES plans, if the value is No. If the value is Yes, it will be propagated to all corresponding plans.

The semantics of a remove expression are as follows:

- The RemovedTargetExpr must refer to a valid AIMSL component; otherwise a static error is raised.
- After removing the RemovedTargetExpr, the parent of the removed component must be a valid AIMSL component or null, otherwise a dynamic error is raised.
- If the remove expression is applied for an ES plan component, the RemovedTargetExpr is logically removed. That means the component is not deleted, but it is marked as a deleted component. Also, the validity period associated with the parent of RemovedTargetExpr should be changed to reflect the new changes that have been made by the remove expression.

Figure 4.28.B shows an example for an *remove* operation that removes a rule, whose id is *rul123* and its schedule id is *sch123*. This schedule is under a protocol, whose id is *pro123*. This *remove* operation will be propagated because the propagation value is Yes.

4.3.2.3 Modify

A *modify* operation might modify a component as a whole or only the values. Figure 4.29.A shows the syntax of the *modify* operation as follows:

```
<xsd:complexType name="modifyDT">
  <xsd:sequence>
    <xsd:element name="value-of" minOccurs="0"/>
    <xsd:element name="ModifyTargetExpr"/>
    <xsd:element name="with"/>
    <xsd:element name="propagation" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

A

```
<modify>
  <ModifyTargetExpr>
  protocol[id="pro123"]//rule[id="rul123"]//event[id="EID123"]
  </ModifyTargetExpr>
  <with>
    +<event id="EID127">
  </with>
</modify>
```

B

Figure 4.29: A: the XML Schema definition of the modify operation. B: an example for a modify operation.

- The *value-of* element determines whether the *modify* operation updates a value or a component.
- The *ModifyTargetExpr* element represents a targeted component in a specific protocol or ES plan.
- The *with* element represents a protocol or ES plan components or a valid value for a protocol or ES plan components.
- the *propagation* values are (Yes or No), and the default value is No. The *modify* operation will not be propagated to the corresponding ES plans, if the value is No. If the value is Yes, it will be propagated to all corresponding plans, if applicable.

4.3.2.3.1 Modify Component. If the *value-of* element is not specified, the *modify* operation modifies one valid AIMSL component with a new valid AIMSL component. The semantics of this form of the *modify* operation are as follows:

- The *ModifyTarggetExpr* must refer to a valid AIMSL component; otherwise a static error is raised. The *ModifyTarggetExpr* is evaluated. The result of this step is either an error or a sequence of component to be modified.

- The *with* element must be a valid AIMSL component; otherwise a static error is raised.
- The result of the modify expression must be a valid AIMSL component.
- If the *modify* operation is applied for a plan, instead of modifying the component targeted by *ModifyTarggetExpr*, a copy of this component will be modified by the *with* element and added as a sibling to the *ModifyTarggetExpr*. Also, the validity period associated with the *ModifyTarggetExpr* should be changed to reflect the new changes that have been made by the *modify* operation.

4.3.2.3.2 Modify the Value of a Component If the *value-of* is specified, the *modify* operation modifies only the value of a valid AIMSL component. The semantics of this form of the *modify* operation are as follows:

- The *ModifyTarggetExpr* must refer to a valid AIMSL component that does not contain another component; otherwise a static error is raised.
- The *ModifyTarggetExpr* is evaluated. The result of this step is either an error or a sequence of components to be modified.
- The *with* element must be a valid value for the *ModifyTarggetExpr* according to AIMSL Schema; otherwise a static error is raised.
- The result of the modify expression must be a valid AIMSL component.

Figure 4.29.B shows a *modify* operation that replaces the event, whose id is *EID123*. This event is under a rule, whose id is *rul123*, and the rule's parent is the protocol, whose id is *pro123*.

4.3.2.4 Activate

The *activate* operation activates an AIMSL *plan, schedule* or *rule* component in a specific plan. This means these components will be ready for the execution process. Figure 4.30.A shows the syntax of the *activate* operation. The semantics of the *activate* operation are as follows:

- The *ActTargetExpr* element must refer to a valid re-playable AIMSL component (*plan, schedule* or *rule*), or to a component containing at least one of these components, such as the *scheduleRules* component.
- As a result to the *activate* operation, the state of the activated component will be transited to the *active* state, and the corresponding triggers will be activated in the system.

<xsd:complexType name="activateDT"> <xsd:sequence> <xsd:element name="ActTargetExpr"/> </xsd:sequence> </xsd:complexType>	<activate> <ActTargetExpr> plan[proid="pro123"]//rule[id="rul123"] </ActTargetExpr> </activate">
A	B

Figure 4.30: A: the XML Schema definition of the activate operation. B: an example for an activiate operation

Figure 4.30.B shows an example for activating a rule, whose id is *rul123*, in a plan, whose *proid* is *pro123*.

4.3.2.5 Deactivate

The *deactivate* operation deactivates an AIMSL *plan, schedule* or *rule* component in a specific plan. This means these components will be off. Figure 4.31.A shows the syntax of the *deactivate* operation, whose semantics are as follows:

- The *DeactTargetExpr* element must refer to a valid re-playable AIMSL component (*plan, schedule* or *rule*), or to a component containing at least one of

these components, such as the *scheduleRules* component.

- As a result to the *deactivate* operation, the state of the deactivated component will be transited to the *inactive* state, and the corresponding triggers will be deactivated in the system.

<xsd:complexType name="deactivateDT"> <xsd:sequence> <xsd:element name="DeacTargetExpr"/> </xsd:sequence> </xsd:complexType>	<deactivate> <DeacTargetExpr> plan[proid="pro123"]//rule[id="rul123"] </DeacTargetExpr> </deactivate>
A	B

Figure 4.31: A: the XML Schema definition of the deactivate operation. B: an example for a dectiviate operation

Figure 4.30.B shows an example for deactivating a rule, whose id is *rul123*, in a plan, whose *proid* is *pro123*.

4.3.2.6 Terminate

The *terminate* operation halts an AIMSL *plan, schedule* or *rule* component in a specific plan. This means these components will be not in use any more. Figure 4.32.A shows the syntax of the *terminate* operation, whose semantics are as follows:

- The *TermTargetExpr* element must refer to a valid re-playable AIMSL component (*plan, schedule* or *rule*), or to a component containing at least one of these components, such as the *scheduleRules* component.

- As a result to the *terminate* operation, the state of the terminated component will be transited to the *terminated* state, and the corresponding triggers will be deleted from the system.

Figure 4.32.B shows an example for terminating a rule, whose id is *rul123*, in a plan, whose *proid* is *pro123*.

<xsd:complexType name="terminateDT"> <xsd:sequence> <xsd:element name="TermTargetExpr"/> </xsd:sequence> </xsd:complexType>	<terminate> <TermTargetExpr> plan[proid="pro123"]//rule[id="rul123"] </TermTargetExpr> </terminate>
A	B

Figure 4.32: A: the XML Schema definition of the terminate operation. B: an example for a terminate operation

4.3.2.7 Fire

The *fire* operation is applying only to the *rule* component in a specific plan. This means the rule's action will be carried out if the rule condition is evaluated to true. Figure 4.33.A shows the syntax of the *fire* operation, whose semantics are as follows:

- The *FireTargetExpr* element must refer to a valid AIMSL rule component in an ES plan.
- As a result to the *fire* operation, the corresponding triggers will be activated regardless the their event.

<xsd:complexType name="fireDT"> <xsd:sequence> <xsd:element name="FireTargetExpr"/> </xsd:sequence> </xsd:complexType>	<fire> <FireTargetExpr> plan[proid="pro123"]//rule[id="rul123"] </FireTargetExpr> </fire>
A	B

Figure 4.33: A: the XML Schema definition of the fire operation. B: an example for a fire operation.

Figure 4.33.B shows an example for firing a rule, whose id is *rul123*, in a plan, whose *proid* is *pro123*.

4.3.3 The AIMQL Replay Query Support

This section presents the AIMQL replay query support to the complex information (skeletal plan and entity-specific plan). Both skeletal plan and entity-specific plan are represented and stored as XML document. Therefore, any XQuery engine could

be used to query them. However, querying entity-specific plans demands a special query operator, which is capable of querying the history at a declarative and high level. This section focuses on the AIMQL replay queries.

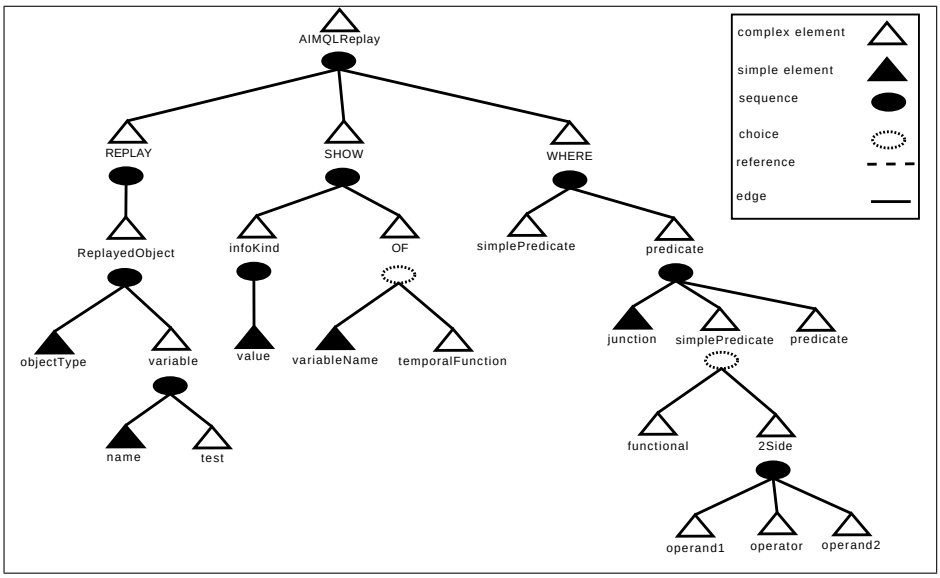

Figure 4.34: The AIMQL replay query structure.

4.3.3.1 The AIMQL Replay Language

The AIMQL replay language is a language that plays over again the history of the entity-specific plans to show the in details the actions that cause changes on the ES plans. At the same time, it considers the skeletal plan specification and application domain information in the query. Figure 4.34 illustrates the XML Schema for the AIMQL replay query structure. The AIMQL replay query statement consists of main three clauses as follows:

- The *REPLAY* clause which indicates the element that is subject to be replayed. As shown in Table 4.2, the special queries (replay queries) are applied only to the plan, schedule and rule elements. As shown in Figure 4.34, the *REPLAY* element has a complex type that consists of a sequence of *ReplayedObject*

elements, which has a complex type that consists of a sequence of elements (*objectType* and *variable*). The *objectType* element is of token type restricted to values (*plan, schedule* and *rule*). The *variable* element has a complex type that consists of a sequence of elements (name, test). The *name* element refers to the variable name, which should be unique. The *test* element is used to specify a condition that restricts the variable to a specific *plan, rule* or *schedule.*

- The *SHOW* clause which determines which pieces of information are to be returned. As shown in Figure 4.34, *SHOW* element has a complex type that consists of a sequence of elements (*infoKind* and *OF*). The *infoKind* element has a complex type that consists of at least one *value* element, which is of type token that is restricted to values (*when*, *why*, *who*, *how*, and *what*). These values are used as an indicator to specify the information kind, which is of interest to the user, as follow:

  (1) the *when* value is an indicator to show the validity period.

  (2) the *why* value is an indicator to show the event that causes the firing, and the condition evaluated in order to execute the rule.

  (3) the *who* value is an indicator to show the actor participating in performing the rule.

  (4) the *how* value is an indicator to show the action carried out.

  (5) the *what* value is an indicator to show the corresponding specification; stored in the skeletal plan; of each component.

  The *OF* element has a complex type that consists of a sequence of elements (*variableName*, *temporalFunction*). The *variableName* element is referring to one of the variables defined under the *REPLAY* clause. The *temporalFuction* element calls one of the temporal function provided by AIMQL, such as

overlaps, meets, first, last, valid, and cast.

- The *WHERE* clause which includes a comparison predicate, which is used to restrict the number of elements returned by the query. The *WHERE* clause eliminates all rows from the result set where the comparison predicate does not evaluate to true. As shown in Figure 4.34, *WHERE* element has a complex type that consists of a sequence of elements(*simplePredicate* and/or *predicate*). The *simplePredicate* element specifies a simple predicate that calls one of the temporal function, or a two side predicate that is consists of two operands connected by an operator. The *predicate* element is used in the case of dealing with composite predicate that is consists of two simple predicates connected by a junction, which is *and* or *or*. The *predicate* element is a recursive element that calls itself in order to support *N* number of composite predicates.

4.3.3.2 Examples: Replay Patterns

This subsections provides several replay query patterns and their corresponding AIMQL replay queries. These patterns covers several situations that shows the capacity of the AIMQL replay queries.

Replay Pattern 1 It is required to retrieve the history of the plan no (@domainEntityID,@protocolID) (X,PID) over the period from TP1 to TP2. In this pattern the variables X, PID, TP1, and TP2 are to be replaced with appropriate values. Figure 4.35 illustrates the AIMQL replay queries for pattern 1. The replay query returns N versions of the plan no. X over the mentioned period.

```
REPLAY PLAN p1
SHOW When OF p1
WHERE p1[@domainEntityID = X and @protocolID = PID]
      and p1.overlaps(valid(TP1,TP2))
```

Figure 4.35: The AIMQL replay query for pattern 1.

Replay Pattern 2 It is required to retrieve the the first version of the plan no (@domainEntityID,@protocolID) (X,PID). In this pattern the variables X, and PID are to be replaced with appropiarte values. Figure 4.36 illustrates the AIMQL replay queries for pattern 2. This replay pattern returns the first version of the plan no. X.

```
REPLAY PLAN p1
SHOW When OF FIRST(p1)
WHERE p1[@domainEntityID = X and @protocolID = PID]
```

Figure 4.36: The AIMQL replay query for pattern 2.

Replay Pattern 3 It is required to retrieve the the last version of the plan no (@domainEntityID,@protocolID) (X,PID). In this pattern the variables X, and PID are to be replaced with appropiarte values. Figure 4.37 illustrates the AIMQL replay queries for pattern 3. This replay pattern returns the last version of the plan no. X. This replay pattern returns the most recent version of the complex information no. (X,PID).

```
REPLAY PLAN p1
SHOW When LAST(p1)
WHERE p1[@domainEntityID = X and @protocolID = PID]
```

Figure 4.37: The AIMQL replay query for pattern 3.

Replay Pattern 4 It is required to retrieve the history of the plan no (X,PID) before executing rule no. R of schedule no S. In this pattern the variables X, and PID, R and S are to be replaced with appropiarte values. Figure 4.38 illustrates the AIMQL replay queries for pattern 4. This replay pattern returns the plan versions, which has no. X and its validity period precedes the validity period of the state executed of rule R.

```
REPLAY PLAN p1
SHOW When OF p1
WHERE p1[@domainEntityID = X and @protocolID = PID] and
        p1.precedes(valid(p1.schedule[@id=S]/rule[@id = R]/state[value = 'executed']))
```

Figure 4.38: The AIMQL replay query for pattern 4.

Replay Pattern 5 It is required to retrieve the history of the schedule no S1 of the plan no (X,PID) when the state of the rule no R of schedule W was ST. In this pattern the variables X, and PID, R, W, and ST are to be replaced with appropriate values. Figure 4.39 illustrates the AIMQL replay queries for pattern 5. This replay pattern returns the versions of Schedule no S1 of the complex information no X, such that the validity of the version overlaps the validity period of the state ST of rule R in schedule W.

```
REPLAY PLAN p1, SCHEDULE p1.schedule[@id = S1] CIS
SHOW When, How, Why OF CIS
WHERE p1[@domainEntityID = X and @protocolID = PID] and
        CIS.overlaps (valid(p1.schedule[@id=S2]/rule[@id=R]/state[value/status = ST]))
```

Figure 4.39: The AIMQL replay query for pattern 5.

Replay Pattern 6 It is required to replay the plans of category no CAT, which was working for more than Y hours. In this pattern the variables CAT and Y are to be replaced with appropriate values. Figure 4.40 illustrates the AIMQL replay queries for pattern 6. This replay pattern returns the versions of the plans of category CAT, whose validity period meets the current time, and whose age is greater than or equal Y hours.

```
REPLAY PLAN p1
SHOW When, How, Why OF p1
WHERE p1[@domainEntityID = X and @protocolID = PID] and
        cast(p1,hour) >= Y
```

Figure 4.40: The AIMQL replay query for pattern 6.

Replay Pattern 7 It is required to replay the plan no (X1,PID1) after the validity period of the state ST of the plan no (X2,PID2). In this pattern the variables X1,

PID1, ST, X2, and PID2 are to be replaced with appropriate values. Figure 5.17 illustrates the AIMQL replay queries for pattern 7. This replay pattern returns the versions of the plan no X1, whose validity period does not precede the validity period of the state ST of the plan no X2.

```
REPLAY PLAN p1,p2
SHOW When, How, Why OF p1
WHERE p1[@domainEntityID = X1 and @protocolID = PID1] and
      p2[@domainEntityID = X2 and @protocolID = PID2] and
      NOT(p1.precedes(valid(p2.state[value/status=ST])))
```

Figure 4.41: The AIMQL replay query for pattern 7.

Replay Pattern 8 It is required to retrieve When and Why was rule A of the schedule S on plan X,PID executed. In this pattern the variables X1, PID1, A, and S are to be replaced with appropriate values. Figure 4.42 illustrates the AIMQL replay queries for pattern 8. This replay pattern returns the versions of the plan no X1, whose validity period does not precede the validity period of the state ST of the plan no X2. This replay pattern returns the versions of the rule R of schedule S of the plan no X, such that the replay period is the period, at which the rule R was executed. The event and the condition evaluation will be shown as well.

```
REPLAY RULE plan[@domainEntityID = X1 and @protocolID = PID1]//schedule[@id=S]/rule[@id=A] R
SHOW When, Why
WHERE R.meet(valid(R.state[value/status=ST]))
```

Figure 4.42: The AIMQL replay query for pattern 8.

Replay Pattern 9 It is required to retrieve How many times was the rule R of the schedule S of the plan (X,PID) executed, and why. In this pattern the variables X1, PID1, R, and S are to be replaced with appropriate values. Figure 4.43 illustrates the AIMQL replay queries for pattern 9. This replay pattern returns the versions of the plan no X1, whose validity period does not precede the validity period of the state ST of the plan no X2. This replay pattern counts the state value executed of

the rule R of the schedule S of the plan no X, and shows the evaluation of the rule R event and the condition at each execution.

```
REPLAY RULE plan[@id=X]//schedule[@id=S]/rule[@id=A] R
SHOW How, Why OF count(R.state[value/status='executed']
WHERE R.meet(valid(R.state[value/status='executed']))
```

Figure 4.43: The AIMQL replay query for pattern 9.

Replay Pattern 10 It is required to retrieve What is the rule R of the schedule S of the plan X1,PID1. In this pattern the variables X1, PID1, R, and S are to be replaced with appropriate values. Figure 4.44 illustrates the AIMQL replay queries for pattern 10. This replay pattern returns the specification of the rule A of the schedule S of the plan X.

```
REPLAY Rule plan[@domainEntityID = X1 and @protocolID = PID1]//schedule[@id=S]/rule[@id=A] R
SHOW What OF R
WHERE R.meet(current time)
```

Figure 4.44: The AIMQL replay query for pattern 10.

4.4 Chapter Summary

This chapter has presented the AIM language that is developed to support the SIM approach and framework. The AIM language is a high-level, declarative, and XML-based language that is divided into three components, AIMSL, AIM ESPDoc model, and AIMQL.

The AIMSL is the AIM specification component that support the formalization process of the complex information as skeletal plans that is represented as XML document. The AIMSL model is based on the ECA rule paradigm with extensions to support temporal events and conditions at the application domain level.

The AIM ESPDoc model is a computer-interpretable model for the entity-specific plan, which consists of four components; *knowledge action*, *domain information*, *de-*

scriptive information and *evolution history*. These four components are supported by the AIM ESPDoc model that is capable of storing the evolution history of the ES plan, as well as the descriptive information regarding the ES plan. The *knowledge action* and *domain information* components of a specific skeletal plan together are utilized to generate *rule* component of the ESPDoc model. The AIM ESPDoc model demands a temporal XML support to be realized.

The AIM language specifies the complex information; the skeletal plans and entity-specific plans as XML document that is to be stored in an XML database. The third component of the AIM language is the AIMQL, which is the AIM query component. AIMQL provides support for manipulating and querying the complex information, and provides special manipulation operations, such as *activate, deactivate* and *terminate* operation, and query capabilities for the complex information.

XML is generally used as a standard for data representation and exchange on the WWW and between heterogeneous systems. Several XML query languages have been developed. The most standard XML query language is XQuery language (Boag et al. 2007). XQuery language is a W3C standard. Because, the AIM language is based on XML: 1) XQuery queries can be used to query the AIM specification, 2) AIM specification can be easily transformed into different format representation. For example, the AIM specification could be transformed to HTML using a stylesheet language, such as XSLT, 3) AIM is also easy to be distributes among heterogeneous systems, 4) ordinary XML tools can be used to facilitate the development of AIM.

The implementation of AIM language demands several extension to the modern DBMSs that support the ECA rule paradigm and XML. The DB triggering mechanism of the modern DBMS should be extended to support 1) the time-based and domain specific events and 2) temporal condition in order to support the AIMSL

language. The XML data model should be extended to support the temporal dimension, which is required to support the AIM ESPDoc model. The AIMQL could be implemented by translating the AIMQL queries into XQuery.

5

AIMS: A Proof-of-Concept System for Managing the Complex Information

This chapter presents a proof-of-concept System called AIMS with focus on its main components. AIMS is an acronym for *A*dvanced *I*nformation *M*anagement *S*ystem. The AIMS system provides 1) a relational database model called *TRME* for executing the AIMSL rules by translating these rules into pure SQL triggers managed by the database management system (DBMS); 2) a temporal XML data model called *TXME* for implementing the AIM entiy-specific plan model using the XML support provided by the modern DBMSs; and 3) an implemention for the AIMQL sub-language based on translating the AIMQL queries into pure XQuery queries.

The chapter is organised as follows: Section 5.1 presents the AIMS structure and the required features that should be provided by a DBMS to be used by AIMS; Section 5.2 discusses the AIMS design at three levels of abstractions, conceptual, logi-

cal, and physical; Section 5.3 presents the *TRME* model for executing the AIMSL rules; Section 5.4 discusses the limitations and performance of the AIMS execution mechanism; Section 5.5 introduces the AIMS method for calculating the expire date of the entity specific plan; Section 5.6 presents the *TXME* model for implementing the AIM entity-specific plan model; Section 5.7 discusses the AIMS method for logging the execution history; Section 5.8 presents the AIMS implementation for the AIMQL sub-language; and Section 5.9 summarizes the chapter.

5.1 AIMS Conceptual Structure and DBMSs support

The AIMS system utilizes the available database management systems (DBMS) as a base for managing the complex information and implementing the AIM language. New sub-systems must be introduced to DBMSs to support the management of the complex information as it is modelled in the SIM approach. The conventional sub-systems of DBMSs must be modified to support advanced features required in real-world situations, such as time events, temporal data management. AIMS adopts the service-oriented architecture based on Web services to provide decentralized management for the complex information. The following sub-sections discuss the functional decomposition of AIMS, whose conceptual structure is shown in Figure 5.1, and the AIMS required features for using a DBMS.

5.1.1 A functional decomposition of AIMS

The main components of AIMS, depicted in Figure 5.1, are:

- the *Complex Information (CI) Manager* that provides the high level management for the complex information (the skeletal plan and entity-specific plan). It supports the main functions of the three plans in the SIM framework. The AIM language is used to communicate with the *Complex Information Manager*.

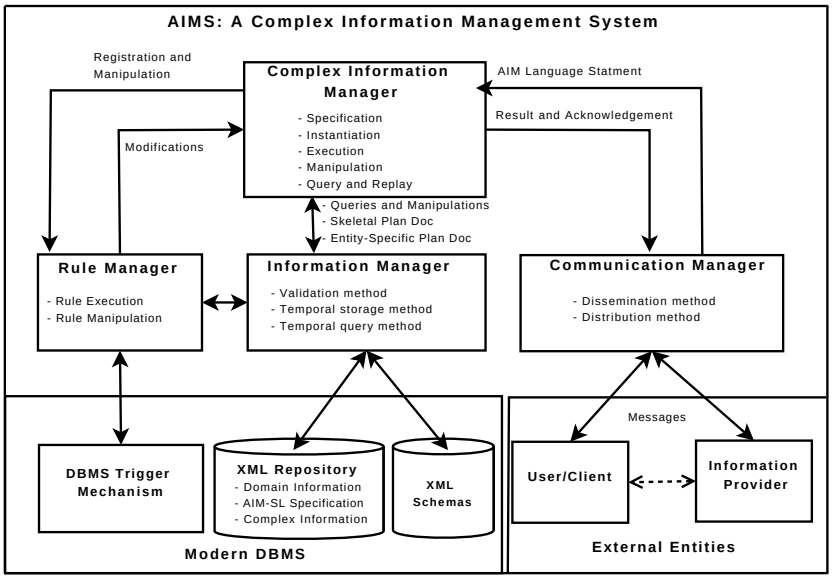

Figure 5.1: AIMS: A proof-of-concept system for complex information management.

- the *Rule Manager* that provides an intermediate model for translating the AIMSL rules existing in a skeletal plan into pure SQL triggers in a corresponding entity-specific plan, plus a method for logging the execution history of the entity-specific plan, and calculating the expire date of the entity-specific plan. The *Rule Manager* manages the execution of the entity-specific plan rules, manipulates them, and extends the DBMS triggering mechanism, e.g., to support temporal triggers. The main functionality of the *Rule Manager* is presented in Sections 5.3. AIMS avoids the unexpected interactions that most likely appear with the growing of the rule base, because 1) the rules are modularized and joined to a specific domain entity instead of specific relation, such as table in the relational database, and 2) the rule manager is able to remove a set of rules according to it objective or scope.

- the *Information Manager* that provides support for managing the skeletal plans and entity-specific plans as XML documents. It develops an intermediate model for extending an XML DBMS to provide temporal support for the

entity-specific plans. The *Information Manager* utilizes an XML DBMS to validate and store the specification of the AIMSL specification (skeletal plan) and the entity-specific plans documents. The *Information Manager* provides an implementation for the AIMSL model and the entity-specific plan model. The main functionalities of this component are presented in Sections 5.2, 5.6, and 5.8.

- the *Communication Manager* that supports the remote access and distributed management to the CI. The communication manager interacts with the external entities, such as users and information provider(s), through messages. The received messages from the external entities embed AIM language statement(s). The Web services related standards, such as WS-Notification (Graham et al. 2004), combined with the DBMS triggering mechanism is utilized to develop the execution process of the CI model. The rule body consists of the elements; *terms, event, condition,* and *action.* The *terms* element maps the general terms, which are used in the *event, condition* and *action* elements of the rule, to specific data items. As shown in Figure 5.1, the general terms are stored as domain information in the XML repository of AIMS. Using WS-Notification, the communication manager subscribes the data items. As soon as the updates to data items become available, the information provider(s) publishes these updates. The communication manager receives the updates. Then the domain information is manipulated according to these updates. The *event, condition* and *action* parts of the rules are translated into triggers, which might be fired, once the general terms associated with these triggers are modified. Only the infrastructure of the decentralized management is in the focus of this research project. Therefore, the *Communication Manager* is not fully implemented in the AIMS system.

- *External Entities* contain the users of the AIMS system and the *Information Provider*, who manages the domain entity information, such as patient healthcare record and supplies the AIMS system by the modification on this information. The communication between the *External Entities* and the AIMS system is to be through messages managed by the *Communication Manager*.

- *Modern DBMS* is to be used as the core of the AIMS system. The main functionality of the *Modern DBMS* is to provide a triggering mechanism and XML data management supports. Both supports are extended by the *Rule Manager* and *Information Manager*.

5.1.2 The Criteria of Selecting a Modern DBMS for AIMS

AIMS utilizes the modern DBMSs to provide the core functionality for the *Information Manager* and the *Rule Manager*. The suitable modern DBMS, which could be used to support the AIMS system, should generally provide a triggering mechanism, Jave stored procedure, XML storage and retrieval. The AIMS required features, which should be supported by the adopted DBMS, could be classified into categories. These categories are *XML Schema, XML Storage, XML Retrieval, XML Update, Triggering Mechanism, Job Scheduler, Temporal Support* and *Web Services*. The available DBMSs are to be evaluated according to these required features to determine the suitable DBMS(s) that could be utilized by AIMS.

The *XML Schema* category includes the features, *validation* and *recursion*. The *validation* feature means the DBMSs provides the ability to register XML Schemas. This feature is required to register the AIMSL schema and the AIM ESDoc model to validate AIMSL specifications and entity-specific plan documents, respectively. The *recursion* feature means that the DBMS support the XML Schema that has recursion. An XML Schema will have recursion if one of its elements referencing

itself, such as in AIMSL schema the *condition* might contain a composite predicate that contains the element *morePredicate* of type composite predicate, more details about the AIMSL schema is provided in Chapter 4.

The *XML Storage* category includes the features, *Size* and *Storage Model Based on*. First the *Size*, the AIMS system system records the execution history of the entity-specific plans. There is limitation in the size of the XML document that could be handled by the modern DBMSs. It is required to have reasonable support to deal with big XML documents. Second the *Storage Model Based on*, the modern DBMSs support the XML storage based on different models, such as relational database (RDB) with Btree index or object-relational database (ORDB). The used model might affect the retrieval performance. For example, using the Btree index enhances the retrieval performance.

The *XML Retrieval* category includes the features, *XQuery Support* and *SQL/XML Support*. The *XQuery Support* is required to support the AIMS implemention for the AIMQL queries. AIMS translates the AIMQL queries into XQuery. Therefore, the adopted modern DBMS should provide an XQuery engine. The *SQL/XML Support* mean that the SQL language is extended to support several XML functions (Andrew and Melton 2002; Sql/Xml 2003).

The *XML Update* category includes the features, *Update Level* and *Standard*. The *Update Level* feature determines at which level the DBMS can update the XML document. The update levels range from document level to node level. The document level means that the update operations are applied only to the whole document. The node level means that the update operations are applied to any node in an XML document. AIMS system demands update support at the node level, as the AIMQL manipulation operations update the AIMSL specification or the entity-specific plans at the node level. The *standard* feature determines whether the

update support is according to the World Wide Web Consortium (W3C) standard or not. AIMS prefers that the update support following the W3C in order to be platform independent.

The *Triggering Mechanism* category includes the features, *Associated with XML Repository*, *XQuer Support* and *SQL/XML Support*. The *Associated with XML Repository* feature means that the triggering mechanism is provided to the XML data. AIMS needs this feature if AIMS deals with domain knowledge stored in XML document. Otherwise, AIMS needs only the triggering mechanism with the relational data. The *XQuer Support* and *SQL/XML Support* features means the triggering mechanism could be specified using XQuery or SQL/XML language. This feature is required if the AIMSL rules are to deal with XML data. In this case, the AIMSL rules should be mapped into XQuery triggers or SQL/XML triggers.

The *Job Scheduler* category includes the features, *Minimum Time Granularity* and *SQL Script Support*. The *Minimum Time Granularity* feature determines the minimum granularity that could be support by AIMS for the AIMSL rules. As explained in Chapter 4, the events of the AIMSL rules support several time granularities ranging from second to year. The *SQL Script Support* feature means that the job scheduler of the DBMS can execute a SQL script. This feature is to be discussed in Section 5.3.

The adopted modern DBMS should provide support for: 1) Java Stored Procedure in order to support the AIMSL advanced actions; 2) basic temporal support, such as dateTime and time stamp data types; and 3) Web Services support in order to be able to receive or send messages.

Most of the modern DBMSs, which provide support for Native XML technology, extend their relational DBMS features to support XML storage and retrieval, such DB2 (Nicola and Linden 2005) and Oracle (Mark Scardina 2004; Zhen Hua Liu

Comparison Features		DB2	Oracle
XML Schema	Validation	Yes	Yes
	Recursion	Yes	No
XML Storage	Size	2G	64K
	Storage Model Based on	RDB + Btree	ORDB
XML Retrieval	XQuery Support	Yes	Yes
	SQL/XML Support	Yes	Yes
XML Update	Update Level	node	node
	Standard	Yes	Not
Triggering Mechanism	Associated with XML Repository	Yes	Yes
	XQuery Support	Yes	Yes
	SQL/XML Support	Yes	Yes
Job Scheduler	Minimum Time Granularity	Minutes	Minutes
	SQL Script Support	Yes	Yes
Java Stored Procedure		Yes	Yes
Temporal Support		Partial	Partial
Web Services		Yes	Yes

Table 5.1: Comparison summary of the support provided by modern DBMSs for the AIMS system

2005). Table 5.1 summarizes our comparative analysis to the support provided to AIMS by the modern DBMSs using the AIMS required features. The comparative analysis is applied to DB2 Express-C version 9.5 and Oracle 10g release 2. As illustrated in Table 5.1 , the most important findings of our comparative analysis are:

- the main drawback of Oracle is the limitation in the size of the XML document, which is too small for an application storing the history;
- DB2 provides support for most of the requirements of AIMS. AIMS is implemented using DB2, Java, and XML technologies, such as XQuery and Web services.

5.2 Conceptual, Logical, and Physical Design of AIMS System

This section presents the design of AIMS storage and functionality (execution mechanism, specification and query language) at three levels of abstractions; conceptual, logical and physical. The implementation method adopted in this research uses the combined application of the Event-Condition-Action (ECA) rule paradigm, a temporal mechanism, advanced DBMS features and XML technologies.

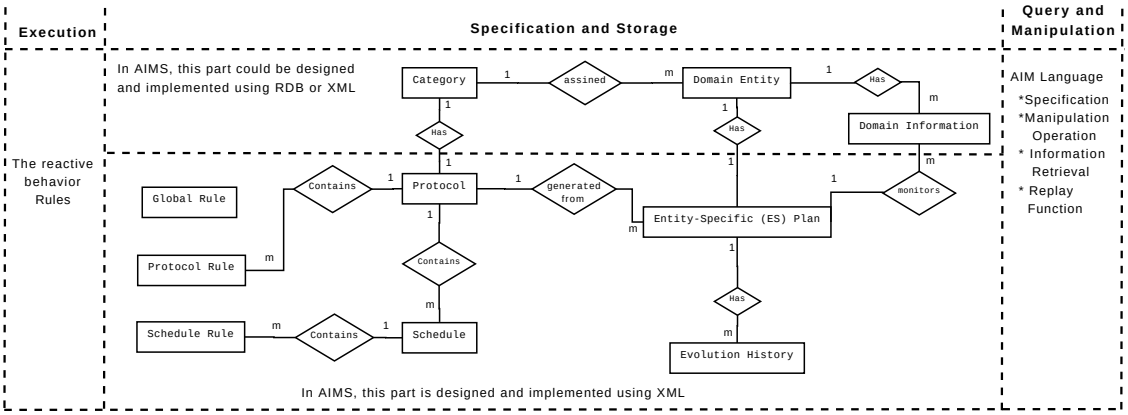

Figure 5.2: The conceptual design of AIMS stotage and functionality

5.2.1 The Conceptual Design

Conceptually, the execution of the complex information is represented as reactive behavior rules that monitor domain information and provide recommendations as an action for detecting specific events of interest. As explained in Chapter 4, the AIM language provides support for specifying, manipulating, and querying the complex information and its execution. The AIMS storage repository at the conceptual level is divided into two main parts, part could be implemented using relational database (RDB) or XML, and another part that is implemented using XML. As shown Figure 5.2:

- The first part is the entities (*domain information*, *domain entity* and *category*) that represent the domain knowledge. In the healthcare domain, these entities are healthcare record, patient, and patient category respectively. This domain knowledge is managed in some domains using RDB and some others using XML database.

- The second part is the entities (protocol, schedule, schedule rules, protocol rules, and global rules) that represent the specification of the complex information using the SIM approach explained in Chapter 3. An entity-specific

(ES) plan is generated from a specific protocol (skeletal plan) and has at least one evolution history. The ES plan is used to monitor the domain information to provide on-line observations and recommendations as soon as an event of interest happened. AIMS implements this part using XML database to support the distributed management of the complex information.

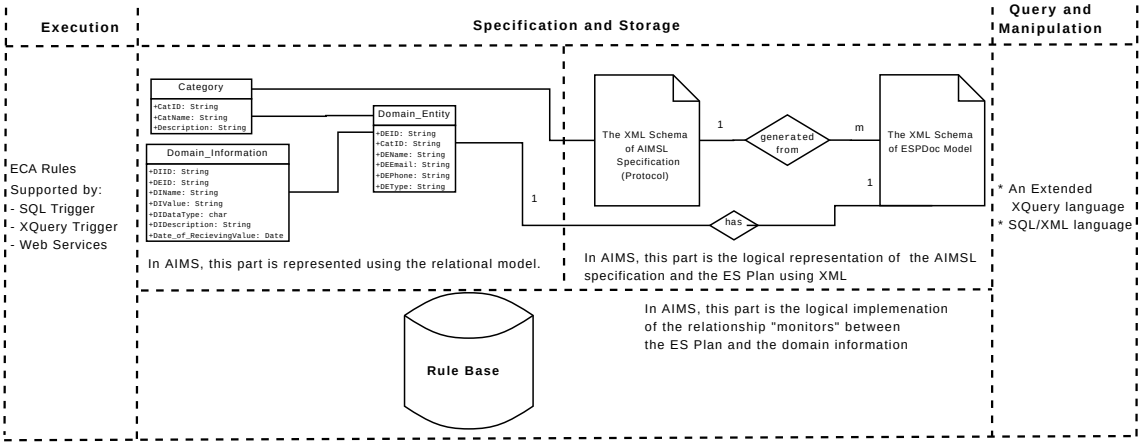

Figure 5.3: The logical design of AIMS stotage and functionality

5.2.2 The Logical Design

At the logical level, the execution of the complex information could be implemented using SQL Triggers (Kulkarni et al. 1999), XQuery Triggers (Bonifati et al. 2002) or Web Services (Cerami 2002). The choice is based on the type of data storage of the domain knowledge (domain information, domain entity and category). The AIM language could be also mapped into XQuery (Boag et al. 2007) or SQL/XML (Andrew and Melton 2002; Sql/Xml 2003). The AIM language is XML-based language. An XML query language should be uased in order to query the protocol (skeletal plan) and/or the ES plan.

The AIMS storage repository at the logical level is divided into three main parts, as shown Figure 5.3:

- The first part is the logical representation of AIMSL specification for the skeletal plans (protocols) and the ESPDoc model for the ES plan documents. Several ES plan documents might generated from a protocol document. A protocol document must be assigned to only one category. The ES plan document provides support for keeping the plan evolution history.
- The second part is the *Rule Base* that contains the rules of the ES plans coded using SQL, XML triggers or Web services. The *Rule Base* is supported by the modern DBMS using a triggering mechanism. However, this support should be extended to cover the AIMSL rules that express real-world situations.
- The third part is the relations, on which the domain knowledge of interest to the complex information is stored. The *domain information* relation stores data items monitored by the ES plan rules. These data items are associated with a specific domain entity, such as the *patient temperature* data item should be associated with a specific patient. The *domain information* relation represents any data items using the attributes (DIID, DEID, DIName, DIValue, DIValueNo, DIDataType, DIDescription). The relation *domain entity* provides a general information about a specific domain entity, such as ID, Name, email, phone, and type. The *type* attribute specifies the type of the entity in the domain, such as in healthcare domain, domain entities could be a patient or clinician. The relation category represents any category in the domain using the attributes (*CatID, CatName* and *Description*).

5.2.3 The Physical Design

At the physical level, The DB2 database management system (DBMS) is utilized to implement AIMS system. The execution of the complex information is implemented using DB2 SQL Triggers, and DB2 SQL/XML language. The DB2 SQL

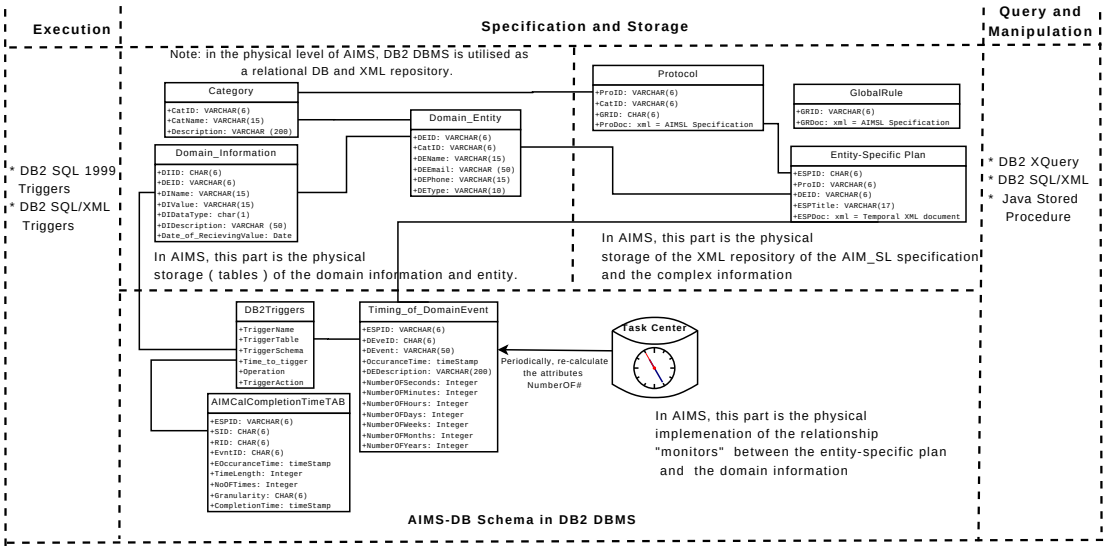

Figure 5.4: The physical design of AIMS stotage and functionality

triggers is chosen because the domain knowledge (*domain information, domain entity* and *category*) in this prototype is stored in relational database. The AIMQL sub-language is mapped into DB2 XQuery.

The AIMS physical database schema is based on the datatypes supported by DB2 database, Figure 5.4 classifies the AIMS database schema into three parts, as the following:

- The first part represents the AIMS XML repository that stores the AIMSL specification of the protocols (skeletal plans) and the ES plans documents. DB2 provides an XML datatype. The attribute of the XML datatype can store an XML document, and the content of this document could be validated using an XML Schema registered in the DB2 database (Nicola and Linden 2005). The DB2 database does not provide support to store and validate a temporal XML document, which is required to support the ES plan documents. It is a demand to extend the DB2 XML datatype to provide support for temporal XML documents. Section 5.6 presents our temporal extension to the XML data model.

- The second part is the physical representation of the ES plan rules and extension to DB2 triggering mechanism. The DB2 task center, which provides support for job schedules, is utilizes to extend the DB2 triggering mechanism to support the AIMSL rules. A pure relational model for extending the triggering mechanism is presented in Section 5.3.

- The third part is the relations, on which the domain knowledge of interest to the complex information is stored. The only difference between this part at the logical and physical level is the used datatypes. The advantage of the *domain information* table is the flexibility to store any kind data items. The *terms* of type element specified in AIMSL language, see Chapter 4, are to be stored in the the *domain information*. Consider as example, the term *patient temperature*, whose second received value is 37.5, will be represented in *domain information* table, as shown in Table 5.2.

DIID	DEID	DIName	DIValue	DIValueNo	DIDataType	DIDescription
DIT131	PAT131	patient temperature	37.5	2	Double	patient temperature

Table 5.2: The *domain information* table.

5.3 TRME: A Model for Translating the AIMSL Rules into SQL Triggers

This section presents an intermediate model, called *TRME*, for translating AIMSL rules into a pure SQL triggers. *TRME* is an acronym for *T*emporal *R*ules *M*ade *E*asy, and implemented using relational database utilities, such as SQL triggers and job scheduler.

5.3.1 TRME Model at Conceptual Level

In AIMSL, the time-based events are classified into two categories of events, absolute time and relative time events. The relative time event is classified into once-off and repetitive event. For more details, the reader is referred to Chapter 4. The main idea behind *TRME* model is to represent the absolute time or the occurrence time of the domain events as tuples of < event id, name, type, description, event occurrence time, <granularity attributes>>. The event occurrence time is the absolute time or the occurrence time of an AIMSL event, and could be greater than or less than the current time.

In temporal data management, there are two types of time, transaction time and valid time (Tansel et al. 1993). The transaction time is the time in which the event happened in the system. The valid time is the time in which the event happened in the real-world. In the *TRME* model, it is assumed that the transaction time and the valid time are equal.

The set of the granularity attributes:

(1) represents the time length towards or afterwards the event occurrence time;

(2) is a set of integer data type attributes, whose values might be negative or positive;

(3) is a set of derived attributes that range from second to years; and

(4) is periodically calculated by subtracting the event occurrence time from the current time and casting the result to a specific granularity, as shown in Figure 5.5.

If the values of the granularity attributes are negative values that means the time length is towards the event occurrence time; otherwise the time length is afterwards

the event occurrence time. Consider as examples for the values of the granularity attributes might be one day, 2 hours, and 3 minutes after *patient admission*, or might be 3 weeks, 2 days, 1 hours, 5 minutes before the *surgery*. Both categories of time-based events, once off and repetitive events, could be represented as triggers, which are triggered after update the corresponding granularity attribute with checking specific predicates that represents a specific time-based event.

DEID	DEVEID	DEName	EOccurrenceTime	SECS	MINS	....	YEARS	Desc
Pat101	DEPA11	Patient Admission	2008-01-14 12:13:52	-9999	-9999	...	-9999	N/A
Pat101	DESU11	surgery	2008-01-20 12:13:52	-999	-9999	...	-9999	N/A

Table 5.3: The initial timing event table for the terms *Patient Admission* and surgery.

As shown on Table 5.3, there are two tuples one for *patient admission* and another one for *surgery* for the same patient (*Pat101*). Consider as examples for time-based once-off events:

(1) on 2 days after *patient admission*; and

(2) on 5 hours before the *surgery*.

The first event could be represent as a trigger, which is triggered after updating the day granularity attribute, and checking the predicates:

- *P1*: the day granularity attribute is equal to the integer value 2;
- *P2*: the other less granularities (from second to hour) should be zero to avoid the repetition (for now, assume the granularity attributes are updated every second); and
- *P3*: the event id is equal AEPA1.

The second event could be represented as a trigger, which is triggered after updating the hours granularity attribute, and checking the predicates:

- *P1*: hours is equal to the negative integer value -5;
- *P2*: the other less granularities (from second to minute) should be zero to avoid the repetition; and
- *P3*: the event id is equal DESU11.

Consider as examples for time-based repetitive events:

(1) every 3 days after *patient admission* for 10 times; and

(2) every 10 hours before *surgery*.

The first event could be represented as a trigger, which is triggered after updating the day granularity attribute, and checking the predicates:

- *P1*: *mod* (ignoreSign(days) , 3) is equal to zero;
- *P2*: the other less granularities (from second to hour) should be zero to avoid the repetition;
- *P3*: the current time is less than the event occurrence time plus (3 * 10 days), this predicate restricts the repetition to 10 times (30 days) only; and
- *P4*: the event id is equal DEPA11.

The second event could be represented as a trigger, which is triggered after updating the hours granularity attribute, and checking the predicates:

- *P1*: *mod*(ignoreSign(hours),10) is equal to zero;
- *P2*: the other less granularities (from second to minute) should be zero to avoid the repetition;
- *P3*: the set of granularity attributes is less than zero, this restricts the repetition until reaching the occurrence time of the *surgery*; and

- *P4*: the event id is equal DESU11.

The function *ignoreSign()* returns the positive the value of the attribute. The episode and absolute time events are also implemented by mapping them into the previous representation and their rules are fired as soon as all the granularity attributes become zero.

5.3.2 DBMS Support for the TRME Model

According to the *TRME* model, the main steps for implementing the AIMSL ECA rules are to:

(1) represent and store the AIMSL events as tuples of the previous representation;

(2) capture the occurrence time of the AIMSL events;

(3) monitor and calculate the time length (the value of granularity attributes); and

(4) translate the AIMSL ECA rules into triggers over the timing event table.

Utilizing the DBMSs to implement the *TRME* model saves the cost of implementing an AIMSL ECA rule execution processor from scratch and extends the modern DBMSs to support the domain-specific and time-based ECA rules.

The tuples of the AIMSL events could be represented and stored in a table, whose schema is (DEID,DEVEID, DEName, OccuranceTimeStamp, Number_of_Seconds, Number_Of_Minutes, Number_Of_Hours, Number_Of_ Days, Number_Of_Weeks, Number_Of_Months, Number_Of_Years, Description). The values of the granularity attributes are calculated using the formula shown in Figure 5.5.

Most of the DBMSs, such as DB2, Oracle, and MS SQL Sever, provide support for scheduling tasks or jobs. This facility is utilized to periodically calculate the value of the granularity attributes as shown in Figure 5.5.

```
CONNECT TO AIMS;
Update aim_TimingDEvent_tab
     SET
        Number_of_Seconds = SECOND ( current timestamp - OccuranceTimeStamp)
       , Number_of_Minutes = MINUTE ( current timestamp - OccuranceTimeStamp)
       , Number_of_Hours = HOUR ( current timestamp - OccuranceTimeStamp)
       , Number_of_Weeks = DAY ( current timestamp - OccuranceTimeStamp) / 7
       , Number_of_Days = DAY ( current timestamp - OccuranceTimeStamp) -
          ( (DAY ( current timestamp - OccuranceTimeStamp) / 7) * 7)
       , Number_of_Months = MONTH ( current timestamp - OccuranceTimeStamp)
       , Number_of_Years = YEAR ( current timestamp - OccuranceTimeStamp);
COMMIT;
```

Figure 5.5: DB2 Task command script for calculating the granularity attributes.

5.3.3 Translating the Terms of the AIMSL Rules

The *term* element in AIMSL is classified into *event* or *element* types. *TRME* translates the *term* element according to its type as explained below. The *term* of type *element* is mapped into a table called, *Domain information*, as shown in Table 6.3. For example, the term *ACR Test Result* is of type element, and consists of *ACR Test Result* as a title, INTEGER as a data type of its value, and its termID is *TO1234*. The term *ACR Test Result*, whose second received value is 37 for patient *PID001*, will be represented in the *domain information* table, as shown in Figure 6.3. This table supports predicates such as " getValue(*TO1234*,3) $>$ 55", which means "check that the third value of the *ACR Test Result* is greater than 55". The terms of type *event* are mapped as shown in Table 5.3. It is assumed that the occurrence time of any event is estimated or given.

DIID	DEID	DIName	DIValue	DIValueNo	DIDataType	DIDescription
TO1234	PID000	ACR Test Result	-99	0	INTEGER	this is an ACR test result
TO1234	PID001	ACR Test Result	37	2	INTEGER	this is an ACR test result
TO1234	PID002	ACR Test Result	-99	0	INTEGER	this is an ACR test result

Table 5.4: The *domain information* table.

5.3.4 Translating the AIMSL Rules into Triggers

This sub-section presents algorithms that are developed to translate the AIMS ECA rule into executable trigger over the timing event table. All the corresponding triggers for the ECA rules are triggered after updating one of granularity attributes and manipulated for each row under specific condition, as explained in details in Algorithm 1 shown in Figure 5.6. The condition clause represents the time-based event.

5.3.4.1 Generate a Trigger

Algorithm 1 translates the AIMSL ECA rule into an equivalent SQL trigger. Algorithm 1 receives as input the AIMSL rule specification, and returns a create trigger statement. The algorithm constructs the create trigger statement. The function *getTriggerName* returns a unique trigger name using the id of the AIMSL rule. The trigging time for all advanced ECA rules is after updating one of the granularity attributes on the timing event table. The functions *getGranularityAttribute* and *getTimingTblName* return a specific granularity attribute used in the rule, and the name of the timing event table, respectively. The generated trigger should be processed for each row in the timing event table, because each row representing a specific event.

The event of an AIMSL rule is represented as a set of predicates in the *when* clause of the generated trigger. If the event type of the AIMSL rule is once-off event, the function *getWhenClauseOE* shown in Figure 5.7, is called to return the equivalent predicates to the once-off event of the rule. Otherwise, if the event type of the AIMSL rule is repetitive event, the function *getWhenClauseRE* shown in Figure 5.8, is called to return the equivalent predicates to the repetitive event of the rule. The function *getWhenClause* is called to construct the predicates equivalent to the

condition part of the rule. All the generated predicates are attached to the *when* clause of the trigger. The action part of the trigger is constructed using the action part of the AIMSL rule.

```
Description : generate a trigger for a specific AIMSL ECA rule
Inputs : r an AIMSL rule specification
Output : ctm create trigger statment
01: ctm = "create trigger " + getTriggerName (r)
02: ctm = ctm + " "+ "after update of "+ getGranularityAttribute( r.event )
03: + " "+ " on " + getTimingTblName()
04: ctm = ctm + " "+ "referencing old as oldrow new as newrow
05: ctm = ctm +" " + "for each row
06: ctm = ctm + " " + "mode db2sql
07: WhenClause = "( "
08: if  getRuleType (r) = "once off" then
09: WhenClause = WhenClause + getWhenClauseOE()
10: else if getRuleType (r) = repetitive then
11: WhereClause = WhenClause + getWhenClauseRE ()
12: end if
13: WhenClause = WhenClause + getWhenClause (r.condition)
14: WhenClause = WhenClause + ") "
15: ctm = ctm + " " + WhenClause
16: ctm = ctm + " " + getTriggerAction(r.action)
```

Figure 5.6: Algorithm 1 getARTrigger.

5.3.4.2 Once Off ECA Rules

Algorithm 2 generates the *when* clause of an once off event. The Algorithm 2 creates two lists of size 7 *gl* and *tll*, one for the granularity attributes and another one for the time length of the corresponding granularity, respectively. The function *getGranularityPosition* returns an integer value p between 0 to 6, which refers to the position of the granularity in the list. If the value of the element *beforeORafter* of the once-off event is *after*, the *timeLengh* is assigned as positive value to cell number p in *tll*, else the *timeLengh* is assigned as negative value to the cell. For each granularity in *gl* list, a predicate is generated. The predicate checks that a granularity attribute is equal to the corresponding time length in the *tll* list. Finally, the algorithm adds another predicate to check that the predicates are evaluated for the AIMSL event, whose ID is equal to the once off event ID.

```
Description : expressing once off advanced event as condition clause
Inputs : oe once off event
Output : WhereClause
01: gl list of all granularity attributes =
02: {"NumberOfMinutes", "NumberOfHours", "NumberOfDays" ,
03: "NumberOfWeeks", "NumberOfMonths", "NumberOfYears"}
04: tll time length list of all granularity = 0,0,0,0,0,0
05: p = getGranularityPosition(oe.granularity)
06: if oe.beforeORafter = "after" then
07: tll[p] = oe.timeLength
08: else tll[p] = - oe.timeLength
09: end if
10: i = 0
12: for each granularity in gl do
13: WhereClause = WhereClause + " ( "+ gl[i] + " = " + tll [i] + " ) "
14: WhereClause = WhereClause + " and "
15: i = i + 1
16: end for
17: WhenClause = WhenClause + " ( AEID = '" + oe.eventID + "' )"
```

Figure 5.7: Algorithm 2 getWhenClauseOE.

5.3.4.3 Repetitive ECA Rules

Algorithm 3 generates the *when* clause of a repetitive time-based event. Algorithm 3 creates one list of size 7 gl, for the granularity attributes. The function *getGranularityPosition* returns an integer value p between 0 to 6, which refers to the position of the granularity in the list. If the value of the element *beforeORafter* of the event is *before*, a predicate will be generated to check that the value of the granularity attribute is less than zero. For the repetition, another predicate is added to check that the result of mod (granularity attribute, the repetition time length) is equal zero. Finally, the algorithm adds another predicate to check that the predicates are evaluated for the advanced event, whose ID is equal to the event ID, to whose value the repetition is calculated.

5.3.4.4 The Condition and Action

In AIMSL, the condition might be a simple predicate or a composite predicate. For more details, the reader is refered to Chapter 4. AIMS implemented only the simple predicate. Using the *TRME* model, the *condition* element is mapped into an SQL predicate, such as the predicate " getValue(*TO1234*,3) $>$ 55" is to be mapped into

```
Description : expressing repetitive advanced event as condition clause
Inputs : re repetitive event
Output : WhereClause
01: gl {list of all granularity attributes} =
02: {"NumberOfMinutes", "NumberOfHours", "NumberOfDays",
03: "NumberOfWeeks", "NumberOfMonths", "NumberOfYears"}
04: p = getGranularityPosition(oe.granularity)
05: if oe.beforeORafter = "before" then
06: WhenClause = WhenClause + " ( "+ gl[p] + " ¡ " 0 ) and "
07: end if
08: WhenClause = WhenClause +
09: " ( mod ( " + gl[p] + " , " + oe.timeLength + " ) )"
10: WhenClause = WhenClause + " and "
11: WhenClause = WhenClause + " ( AEID = '" + oe.eventID + "' )"
```

Figure 5.8: Algorithm 3 getWhenClauseRE.

"(INTEGER(newrow.ValueNO) = 3) AND (INTEGER(newrow.DIValue) > 55)", where *newrow* is a SQL variable referring to the new updated tuple.

The action of an AIMSL rule might be of a procedural action, such as send email, or an AIMQL operation, such as add rule or terminate rule. For more details, the reader is referred to Chapter 4. We have extended the DBMS triggering mechanism using a Java stored procedural to send an email as an action attached with a SQL trigger.

The SQL triggers do not allow any SQL data definition statement (DDL), such as create or drop triggers, to be a part of a SQL trigger. The reason is that the DDL statements enforce the DBMS to commit after executing a DDL statement. Executing a commit statement within the SQL trigger action violates one of the database transaction properties, which is atomic transaction. The database transaction, by definition, must be atomic. Atomic means the work units performed in a database must be completed in their entirety or take no effect whatsoever (Elmasri and Navathe 2003). The SQL trigger is invoked by a database operation, which is not yet committed. Therefore, it is not allowed to execute a commit statement within uncompleted transaction.

In order to implement the AIMSL actions that issue AIMQL operations, such as add or terminate rules, we have adopted the method developed in (Dube 2004). This

method creates a network socket to connect the AIMS database with a *Listener*, message processor, that is outside the database. The *Listener* receives messages from AIMS to create or drop triggers. Using this method, the SQL trigger, which is an implementation to an AIMSL rule, can create or drop another trigger as a part of its action by sending a message to the *Listener* through a Java stored procedure. This way logically does not violate the atomic property of the database transaction.

5.4 The AIMS Execution Mechanism: Limitations and Performance

The AIMS execution mechanism is based on translating the AIMSL rules into a pure SQL triggers over the Timing Event Table using the *TRME* model. The DBMS is to be in charge of managing these SQL triggers.

5.4.1 Limitations

The limitations of the AIMS execution mechanism are classified into: 1) granularity limitations and 2) limitations on the maximum number of triggers.

Granularity limitations are: 1) in most of the DBMSs, the minimum granularity for the job (task) repetition period supported by the job scheduler is minute. This granularity limitation means that the AIMS rules based on the second granularity are practically not supported; and 2) as minimum granularity used as the load on the system increase. For example, supporting the minute granularity means that the job scheduler should run every minute, but if the rules specified at the day granularity, the job scheduler should run every 24 hours.

Limitations on the maximum number of triggers. In the DBMSs, there are several limitations and restrictions on the trigger-based applications (Ceri et al. 2000). We

have classified the limitations on the maximum number of triggers into the following categories:

(1) the total number of triggers per table. There is a limitation on the maximum number of triggers that could be created on a specific table, some DBMS supports up to 300 triggers per table, such as DB2;

(2) The total number of concurrent triggers. The DBMSs have a limitation on multiple triggers that are activated at same time. This limitation based on the buffer size and the complexity of the triggers; and

(3) the number of levels for nested triggers. Triggers are nested when a trigger performs an action that initiates another trigger. There is a limitation on the maximum number of levels supported for the nested triggers, this levels could be in some systems up to 128 level.

Once the job scheduler updates the granularity attributes in the timing event table, all the triggers created over the table will fire and become part of the update transaction that commits after processing all the fired triggers. Consequentially, the above limitations means that using one timing event table, on which all the SQL triggers are to be created, is performance problem and also critical limitation on the maximum number of triggers that could be created or managed at the same time.

5.4.2 Overcoming the Limitations and Enhancing the Performance

The limitations discussed in the previous sub-section are overcome by AIMS as discussed in the next paragraphs.

Overcoming the limitation on nested triggers. The AIMS execution mechanism is not affected by this limitation because the triggers corresponding to the AIMSL

rules do not modify the timing event table, which is updated only by the job scheduler. That means there is no nested triggers.

Overcoming the limitation on the maximum number of triggers per table. The AIMS system overcomes this limitation by performing horizontal fragmentation on the timing event table using for example the domain entity ID, such as patient ID. Therefore, several timing event tables will be in use. Logically, the maximum number of triggers, which could be created on the timing event table, is increased.

Overcoming the limitation on the maximum number of concurrent triggers. The AIMS system overcomes partially this limitation by performing database tuning and job scheduler time slicing. The goal of database tuning is to maximize use of the system resources to perform work as efficiently and rapidly as possible. Slicing the supported granularity among the job scheduler is utilized in AIMS to reduce the number of concurrent triggers. For example, assume that the minimum supported time granularity is an hour and there are three job schedulers *JS1*, *JS2* and *JS3* for updating the timing event tables *TET1*, *TET2*, *TET3* receptively. The time could be sliced as follows: *JS1* runs in the first 20 minutes, *JS2* runs in the second 20 minutes, and *JS3* runs in the third 20 minutes. That leads to fire separately the triggers attached with each timing event table. Therefore the number of concurrent triggersis reduced. Currently, the time slicing in AIMS is made manually.

5.5 AIMS Method for Calculating the Expire Date of the Entity-Specific Plan

The AIMS system provides a pure SQL method for calculating the expire date of the entity-specific plan. The entity-specific plan contains several types of rules, time-based rules and non-time-based rules that based on domain-specific events, such

as test result received. The time-based rules are classified into absolute time event rules, such as on May 23, 2008 do something, or relative time event, two hours after patient admission do something. The relative time event rules are classified into two type once-off rules and repetitive rules. These rules are captured and specified using the AIM specification component. For more details, the reader is refered to Chapter 4.

The AIMS system calculates the duration of each rule in an entity-specific plan. The maximum duration time represents the expire date of the entity-specific plan. While AIMS is registering an entity-specific plan, AIMS calculates the duration of all the rules registered for this plan, and determines the initial expire date of the plan.

Figure 5.9 shows the four rules of the entity-specific plan, number ESP131, that contains one schedule containing four rules. Rule1, Rule2, and Rule3 are time-based rules. Rule3 is an absolute time event rule. Rule1 and Rule2 are relative time event rules of type repetitive and once-off rules, respectively. Rule4 is a domain-specific event, which is on receiving the ACR test. The AIMS system assumes that the occurrence time of the domain-specific events are pre-determined (given) or could be estimated.

Rule1 : 4 minutes After of the patient admission for 10 times do something.
Rule2 : 1 day Before the operation for do something.
Rule3 : On 2008-01-15 10:05:00 do something.
Rule4 : On ACR test received do something.

Figure 5.9: The rules of the entity-specific plan, number ESP131.

While AIMS is registering an entity-specific plan, for each registered rule a tuple is inserted in the *AIMCalCompletionTimeTAB* table, as shown in Table 5.5. The table consists of 9 attributes:

- ESPID is an attribute representing the entity-specific plan ID

- SID is an attribute representing the schedule ID
- RID is an attribute representing the rule ID
- EvntID is an attribute representing the event ID, on which the rule is based
- EOccurrenceTime is an attribute representing the occurrence time of the event
- TimeLength is an attribute representing the time length before or after the event. The negative values refers to *before*, and the positive value refers to *after*. This value could be zero in the case of absolute time event rules and domain-specific event rules.
- NoOFTimes is an attribute representing the number of repetition for the rule. This value has a value only with the repetitive rules.
- Granularity is an attribute representing the granularity of the time length, which could be second, minute, hour, day, week, month, or year.
- CompletionTime is a derived attribute representing the expire date of the rule. The CompletionTime attribute is derived using this SQL formula (EOccurrenceTime + (TimeLength * NoOFTimes) Granularity). For Rule1, the formula is (2008-01-14 12:13:52 + (4 * 10) MINUTE).

ESPID	SID	RID	EvntID	EOccurrenceTime	TimeLength	NoOFTimes	Granularity	CompletionTime
ESP131	Sch1	Rule1	PAD131	2008-01-14 12:13:52	4	10	MINUTE	2008-01-14 12:53:52
ESP131	Sch1	Rule2	OPE131	2008-01-16 12:13:52	-1	1	DAY	2008-01-15 12:53:52
ESP131	Sch1	Rule3	ABS131	2008-01-15 10:05:00	0	0	SECOND	2008-01-15 10:05:00
ESP131	Sch1	Rule4	ACR131	2008-01-16 12:13:52	0	0	SECOND	2008-01-16 14:13:52

Table 5.5: The AIMS table assisting in calculating the expire date of the entity-specific plan.

The expire date for any entity-specific plan or one of its schedule is the maximum completion time of its rules. This simple logic is implemented using a pure SQL query. Figure 5.10.A shows a group-by query selecting the maximum Completion-Time of the rules that belong to the entity-specific plan 'ESP131'. The result of

A
SELECT MAX (CompletionTime) **FROM** AIMCALCOMPLETIONTIMETAB **WHERE** ESPID = 'ESP131' **GROUP BY** ESPID
B
SELECT MAX (CompletionTime) **FROM** AIMCALCOMPLETIONTIMETAB **WHERE** ESPID = 'ESP131' and SID = 'Sch1' **GROUP BY** ESPID, SID
C
SELECT ESPID, SID, **MAX** (CompletionTime) **FROM** AIMCALCOMPLETIONTIMETAB **GROUP BY** ESPID, SID

Figure 5.10: The SQL Query for calculating the expire date.

this query is the expire date of the plan 'ESP131'. If the where-clause is removed from the group-by query, the query will return the expire date of all entity-specific plans registered in the system. Figure 5.10.B shows a group-by query selecting the maximum CompletionTime of the rules that belong to the schedule 'Sch1' of the entity-specific plan 'ESP131'. The query returns the expire date of the schedule. If the where-clause is removed from the group-by query, as shown in Figure 5.10.C, the query will return the expire date of all schedules in the all plans. The entity-specific plan is dynamically changing over time by adding or removing rules. After any modification in the entity-specific plan, the expire date of the plan must be re-calculated.

5.6 TXME: A Temporal XML Data Model for implementing the AIM ESPDoc Model

This section presents a temporal XML data model, called *TXME*, for implementing the AIM ESPDoc model. *TXME* is an acronym for *T*emporal *X*ML *M*ade *E*asy. The *TXME* model is consistent and compatible with both XML Schema and the XML data model. Consequentially, the XML storage and retrieval support provided by the modern DBMSs could be utilized, as it is, to store and retrieve the AIMSL

specification and entity-specific plan documents.

The modern DBMSs, such as DB2 and Oracle, provide partial support, such as date and time data-types, for the temporal data management, as shown in Table 5.1. The modern DBMSs provide an XML data-type that is used to extend their relational database to store XML documents (Mark Scardina 2004; Zhen Hua Liu 2005; Chen et al. 2006; Nicola and Linden 2005). This XML data-type follows the W3C XML data model (Bray et al. 2008). In order to re-use the XML support provided by the modern DBMSs, the temporal extensions should be consistent and compatible with the XML data model.

The XML data model is a tree structure that consists mainly of two types of element *simple element* and *complex element* (Bray et al. 2008). The *simple element* is an element that contains a text value only. The *complex element* is an element that contains other *simple element* and/or *complex element*. The XML data model does not provide support for the temporal relationships between the elements. For examples, a *simple element* might contain the value *V1* at a specific time point, or an element was a child of a *complex element* at a specific time period. The *complex element* might contains attributes, which are pairs of attribute names *ai* and attribute values *Ai*. These temporal relationships are the basic requirements for realizing the AIM ESPDoc model.

The *TXME* model extends the XML data model with the ability to define temporal elements. The temporal element is an element that varies over time. The *TXME* data model could be applied for any conventional XML Schema to generate a temporal Schema, which could be used to validate a temporal XML document. Any instance of the *TXME* model is a temporal XML document that is well-formed XML document and/or a valid XML document, which is defined as a well-formed XML document and conforms to the rules of a Document Type Definition (DTD) or

an XML Schema (XSD). Therefore, the *TXME* model is consistent and compatible with both XML Schema and the XML data model.

In *TXME* model, the time-varying attribute is represented as a time-varying element. On the other hand, the *TXME* model does not support temporal (time-varying) attributes. The temporal elements are classified, according to its content, into two categories, *time-varying simple* element and *time-varying complex* element. In the following sub-sections, the formal definitions for the two categories of the temporal elements are discussed.

(A)
<SElement Attributes< simpleValue </SElement>
(B)
<SElement startTime=ST endTime=ED Attributes> <value startTime=ST1 endTime= ET1> simpleValue</value> <value startTime=ST2 endTime= ET2>simpleValue</value> . . . <value startTime=STn endTime= ETn> simpleValue </value> </SElement>
Such the following temporal constrains: 1) STn <= ETn 2)STn = ETn-1, 3) STn-1 < STn, 4) ETn-1 < ETn, 5) ETn = Now, Now refers to the current time 6) ST = min (ST1, ST2, . . .,STn)= ST1 and 7) ET = max (ET1, ET2,. . . ,ETn)= ETn. 8) 1,2,,n refers different time point.

Figure 5.11: (A) The structure of an XML simple element. (B) The time-varying simple element structure and temporal constrains.

5.6.1 Time-Varying Simple Element

The *time-varying simple* element is an element that has only text node, whose value varies over time, as shown in Figure 5.11.A, which illustrates the structure of an XML *simple element*, whose value is of simple data type such as string or integer. Figure 5.11.B illustrates the *TXME* model for *time-varying simple* element, and depicts the formal definition for the structure and temporal constrains for the model of the *time-varying simple* element.

5.6.1.1 Structure

The *TXME* model for a *time-varying simple* element is different from XML data model for a *simple element* in the point that the *time-varying simple* element can hold multiple elements called *value* that hold the *simple element* value varying over time. Each *time-varying simple* element and *value* element has a validity period that presents the period, in which the element or the value was/is valid, as shown in Figure 5.12.B. The validity period is represented as two attributes, *startTime* and *endTime*, that represents the start time and end time of the validity period, respectively. In the *TXME* model, the *time-varying simple* element might contain a non-temporal *complex element*(s), which will not affect the *time-varying simple* element temporal constrains.

5.6.1.2 Temporal Constrains

Figure 5.11.B illustrates several temporal constrains that determine how to 1) associate a new value to a *time-varying simple* element and 2) adjust the validity period of the *time-varying simple* element. For the same validity period, the start time at any time point should be less than or equal to the end time. Assume the value is changed at time point *n-1*, this means: A) the value of the *endTime* attribute of the current value is *ETn-1*; and B) a new *value* element, whose *startTime* is *ETn-1* and *endTime* is *Now* will be added.

The start time and end time of any validity period of the *value* element at time point *n* is greater than the start time and end time of the validity period of the *value* element at time point *n-1*, assuming there is no changes for the same *simple element* within the supported time granularity. The validity period of the *time-varying simple* element consists of the minimum start time and the maximum end time of all validity periods attached with the elements called *value*. According to

the temporal constrains 3 and 4, the minimum start time is the start time of the validity period attached with the first value, and the maximum end time is the end time of the validity period attached with the last value.

(A)

```
<status>executed</status>
```

(B)

```
<status startTime="2008-01-14T13:25:18" endTime="Now">
    <value startTime="2008-01-14T13:25:18" endTime="2008-01-14T14:25:19">registered</value>
    <value startTime="2008-01-14T14:25:19" endTime="NOW">executed</value>
</status<
```

Figure 5.12: (A) An example for an XML simple element. (B) An example for a TXME time-varying simple element.

5.6.1.3 An Example

Figure 5.12.A illustrates an example for a simple element, called *status*, that represents the status of an AIMSL rule. The status of an AIMSL rule are such as generated, registered, and executed. The status of the AIMSL rule varies over time. Therefore, the element status is a *time-varying simple* element. Figure 5.12.B illustrates the element status as *time-varying simple* element.

The temporal constrains shown in Figure 5.11.B validate the semantic of the temporal data. Figure 5.12.B illustrates the following:

- the start time of the value "registered" is less than its end time;
- the start time of the value "executed" is equal the end time of the value "registered";
- the *NOW* value represents the current time;
- the start time of the status element is the minimum start time belongs to its values, which is "2008-01-14T13:25:18";
- the end time of the status element is the maximum end time belongs to its values, which is the value *NOW*; and

- the element, whose end time is equal *NOW*, is currently valid element.

5.6.2 Time-Varying Complex Element

The *time-varying complex* element is an element that has sub-elements, which might be simple or complex, and temporal or non-temporal. The *time-varying complex* element might have attributes. Figure 5.13 illustrates the *TXME* model for the *time-varying complex* element, and depicts the formal definition for the structure and temporal constrains for the model of the *time-varying complex* element.

<CElement startTime=ST endTime=ED Attributes>
<nonTemporalNode-1 . . .>. . .< /nonTemporalNode-1>
<nonTemporalNode-2 . . .>. . .< /nonTemporalNode-2>
<nonTemporalNode-n . . .>. . .< /nonTemporalNode-n>
<temporalElement-1 startTime=ST1 endTime= ET1 . . .> . . . </temporalElement-1>
<temporalElement-2 startTime=ST2 endTime= ET2 . . .> . . . </temporalElement-2>
<temporalElement-n startTime=STN endTime= ETn . . .> . . . </temporalElement-n>
</CElement>

Such the following temporal constrains:
1) ST = min (ST1, ST2,. . .,STn) and
2) ET = max (ET1, ET2,. . .,ETn)
3) 1,2,. . .,n refers to different element.

Figure 5.13: The time-varying complex element structure and temporal constrains.

5.6.2.1 Structure

The *TXME* model for the *time-varying complex* element is different from XML data model for a *complex element* in the point that the *time-varying complex* element has a validity period, which is represented as two attributes, *startTime* and *endTime*. The validity period presents the period, in which the element was/is valid. The *time-varying complex* element consists of at least one temporal element, which might be a *complex* or *simple time-varying* element.

5.6.2.2 Temporal Constrains

As illustrated in Figure 5.13, the validity period of the *time-varying complex* element consists of the minimum start time and the maximum end time of all validity periods

attached with the temporal elements. There is no temporal relationship between the validity periods of the different temporal elements, which are children of a time-varying complex element.

5.6.2.3 An Example

Figure 5.14 illustrates an example for a *time-varying complex* element, an AIMSL rule element at the run time. The *rule* element is a temporal element consisting of a time-varying complex element, called *state*, and a non-temporal element, called *triggers*. The *state* element consists of *time-varying simple* elements, called *value*. As shown in Figure 5.14, the start time of each *value* element *vi+1* is equal the end time of the the value element *vi*, such that *vi* precedes *vi+1*.

The start time of the *state* element is the minimum start time of its *value* elements, and the end time of it is the maximum end time of its *value* elements. The validity period of the *rule* element is equal to the *state* element validity period because the *state* element is the only temporal element under the *rule* element.

```
<rule IDREF="rul1" startTime="2008-01-14T12:13:29" endTime="NOW">
  <state startTime="2008-01-14T12:13:29" endTime="NOW">
    <value startTime="2008-01-14T12:13:29" endTime="2008-01-14T13:25:18">
      <status>generated</status>
    </value>
    <value startTime="2008-01-14T13:25:18" endTime="2008-01-14T14:25:19">
      <status>registered</status>
    </value>
    <value startTime="2008-01-14T14:25:19" endTime="2008-01-15T08:25:19">
      <status>executed</status>
      <event>time-based rule fire when the plan is 2-hours old</event>
      <action>it sent an email</action>
    </value>
    <value startTime="2008-01-15T08:25:19" endTime="NOW">
      <status>terminated</status>
    </value>
  </state>
  <triggers>
    <trigger> </trigger>
  </triggers>
</rule>
```

Figure 5.14: An example for a time-varying complex element, an AIMSL rule of an ES plan.

5.7 AIMS Method for Logging the Execution History of the Entity-Specific Plan

This section discusses the support provided by the *TXME* model for realizing the AIM ESPDoc model, and the method used to log the execution history of the ES plan.

5.7.1 The TXME Support for the Entity-Specific Plan Model

In *TXME* data model, the temporal XML document is a well-formed XML document, such that its root element is a temporal element, whose validity period contains all the validity periods existing in the document. Therefore, the temporal XML document is consistent and compatible with both XML Schema and the XML data model. Consequentially, the TXME model extends the XML datatype provided by the modern DBMSs, such as Oracle (Mark Scardina 2004) and DB2 (Nicola and Linden 2005).

The ESPDoc model for the ES plan document is a temporal XML-based model. AIMS utilizes the *TXME* model to realize the AIM ESPDoc model, which is specified using an XML Schema that follows and obeys the semantic rules of the TXME model. This XML Schema is to be registered in DB2 database. The ES plan documents stored in the *entity-specific plan* table, which is shown in Figure 5.4, is to be validated against the XML Schema of the AIM ESPDoc model.

5.7.2 Logging the Plan Execution History

Each Rule in the ES plan is translated into DB2 SQL/XML trigger. The generated SQL/XML trigger contains the logic of the corresponding rule plus a procedure for logging the changes made by executing the corresponding rule. AIMS system

implements the semantic of the TXME model using XML update statement. The changes might be under the *state* element of the rule, which consists of several *value* elements.

```
<value startTime="{current Time Stamp}" endTime="{NOW}">
   <status>executed</status>
   <event>{the event evaluation of the rule}</event>
   <condition>{the condition evaluation of the rule}</condition>
   <action>{the action logic of the rule}</action>
</value>
```

Figure 5.15: The new *value* element.

If a trigger is fired and its condition is evaluated to ture, the action of the trigger applies the business logic and logs the changes made by this business logic as one transaction. The reader is referred to review the XML Schema of the ESPDoc model in Chapter 4. The changes are:

(1) Replace the value of *endTime* attribute under the *value* element, whose end time is *NOW*, by the current Time Stamp,

(2) Add the new *value* element shown in Figure 5.15. The {current Time Stamp} is the current system time that is, in this case, equal to the end time of previous *value* element, see step 1. The {NOW} is an AIMS system value, which means at any time point the present time without need to modify its value to reflect the actual current time-stamp. The {the event evaluation of the rule} is the actual evaluation of the event in the run-time, each *event* element contains a *description* element, which is modified by the actual values of the run-time, such as "this is the day *4* of patient admission, the rule fired every two days". The {the condition evaluation of the rule} is the the actual evaluation of the condition, such as "At the evaluation time, the ACR test result was 60, which is greater than 55, the condition was evaluated to true". As well as, the {the

action logic of the rule} is the description of the action.

(3) if the rule applies an operation, such as terminate, add, or deactivate, on another rule, these operations is logged as using the same logic of step 1 (replace) and 2 (add), in which a new *value* element is added under the *state* element, but instead of the <status>*executed*</status>, it could be <status>*terminated*</status>.

5.7.3 An Example

Figure 5.16 illustrates an example for an ES plan rule, which fires every 12 hours if the test result is greater than 55. The steps 1 and 2 are shown with the last two *value* elements shown in Figure 5.16. The value of the *endTime* attribute is replaced with the current time at the execution, which was *2008-03-24T00:26:09*. Then the new *value* element recording the execution history at that time is added with the validity period (*2008-03-24T00:26:09*,*NOW*).

The reason of executing the rule and the rule action are recorded with the rule state values, as shown in Figure 5.16. For example, the last execution happened because the test result was 86, which is greater than 55. The elements, whose *endTime* attribute is equal *NOW*, are the current valid element of the rule. The other elements were valid within their validity period.

5.8 Translating AIMQL Queries into XQuery

This section presents the XQuery templates corresponding to the AIMQL replay language and presents XQuery scripts of some AIMQL replay patterns presented in Chapter 4. As discussed, the *TXME* model support the realization of the AIM ESPDoc model and the process of keeping the execution history of the ES plan. The *TXME* model is fully compatible with the XML data model supported by

```
<rule IDREF="rul5" startTime="2008-01-14T12:13:29" endTime="NOW">
  <state startTime="2008-01-14T12:13:29" endTime="NOW">
    <value startTime="2008-01-14T12:13:29" endTime="2008-01-14T13:25:18">
      <status>generated</status>
    </value>
    <value startTime="2008-01-14T13:25:18" endTime="2008-01-15T00:25:19">
      <status>registered</status>
    </value>
    <value startTime="2008-01-15T00:25:19" endTime="2008-01-16T00:25:33">
      <status>executed</status>
      <event>time-based rule fires every 12 hours if the test value is gt 55</event>
      <condition>the condition (test value is greater than 55) is true, because the test value was 83</condition>
      <action>it sent an email</action>
    </value>
    ...
    <value startTime="2008-03-23T00:26:10" endTime="2008-03-24T00:26:09">
      <status>executed</status>
      <event>time-based rule fires every 12 hours if the test value is gt 55</event>
      <condition>the condition (test value is greater than 55) is ture, becuase the test value was 81</condition>
      <action>it sent an email</action>
    </value>
    <value startTime="2008-03-24T00:26:09" endTime="NOW">
      <status>executed</status>
      <event>time-based rule fires every 12 hours if the test value is gt 55</event>
      <condition>the condition (test value is greater than 55) is ture, becuase the test value was 86</condition>
      <action>it sent an email</action>
    </value>
  </state>
    ...
</rule>
```

Figure 5.16: An example for an ES plan rule.

the modern DBMS. Consequentially, the AIM ES plan document, which is based on the *TXME* model, could be queried using the XQuery language. Therefore, translating the AIMQL replay queries into pure XQuery scripts is adopted as an implementation method for the AIMQL language.

The AIMQL replay query is a declarative query, which means that the user does not need to know the structure of the complex information (Skeletal plan and ES plan). The translator knows the structure (elements and attributes) of the ES plan, Skeletal plan and the *domain information* table. The translator generates the equivalent XQuery that allows new XML document to be constructed as a result of the AIMQL replay queries. Each part of an AIMQL replay query is translated into its corresponding XQuery. The AIMQL replay query consists of *REPLAY*, *SHOW*, and *WHERE*, for more details the reader is referred to Chapter 4.

5.8.1 The XQuery template for the AIMQL Replay Variables

The AIMQL variables are defined in the *REPLAY* clause to access specific elements that are subject to be replayed. These elements must be of type *plan*, *schedule*, or *rule*. The variables might be restricted using a specific condition, which might be specified in the *REPLAY* or *WHERE* clauses. If the variable appears in the *SHOW* clause that means there is a need to set up an iteration through the element associated to the variable. For example, if the variable *P1* of type *plan*, that need to set up an iteration through the *plan* element and its sub-elements, such as *schedule* and *rule*. Setting up iterations through these sub-element is implicitly demand.

The XQuery provides the *FOR* clause support the iteration. Consequentially, the variables appear in the *SHOW* clause are defined within an XQuery *FOR* clause. However, if variables do not appear in the *SHOW* clause, its expressions in the AIMQL replay query are defined within an XQuery *LET* clause, which binds the variable to specific value. For example, the AIMQL replay query for pattern 7 defines two variables of type *plan*, *p1* and *p2*, as shown in Figure 5.17. The variable *p1* is used in the *SHOW* clause. The variable *p2* is used only in one expression in the *WHERE* clause. The XQuery template for the variables *p1* and *p2* are shown in Figure 5.18.

```
REPLAY PLAN p1,p2
SHOW When, How, Why OF p1
WHERE p1[@domainEntityID = X1 and @protocolID = PID1] and
      p2[@domainEntityID = X2 and @protocolID = PID2] and
      NOT(p1.precedes(valid(p2.state[value/status=ST])))
```

Figure 5.17: The AIMQL replay query for pattern 7.

XQuery uses functions, such as *doc* and *collection*, to access XML documents from within a query (Walmsley 2007). In DB2, an XQuery can obtain input data by calling a function named *db2-fn:xmlcolumn* with a parameter that identifies the

```
let $vaildST :=
db2-fn:xmlcolumn('AIM_ESPlan_TAB.ESPDOC')
//Plan[@domainEntity_ID=X2 and @protocol_ID=PID2]/state[value/status=ST]/xsd:dateTime(@startTime)

let $vaildET :=
db2-fn:xmlcolumn('AIM_ESPlan_TAB.ESPDOC')
//Plan[@domainEntity_ID=X2 and @protocol_ID=PID2]/state[value/status=ST]/xsd:dateTime(@endTime)

for $p1 in
db2-fn:xmlcolumn('AIM_ESPlan_TAB.ESPDOC')//Plan[@domainEntity_ID=X1 and @protocol_ID=PID1]
```

Figure 5.18: The XQuery template for the variables *p1* and *p2*.

table name and column name of an XML column in a DB2 table (Chen et al. 2006). Figure 5.4 shows the tables representing the AIMS XML repository. These tables are AIM_ESPlan_TAB, AIM_Protocol_TAB, and AIM_GlobalRules_TAB.

As shown in Figure 5.18, the equivalent of the AIMQL replay variable *p1* and of *p2* type *plan* are:

- for *p1*, the XQuery variable $p1 that is defined within a *FOR* clause, which sets up an iteration over the *plan* element, whose attributes @domainEntity_ID and @protocol_ID are equal the values *X1* and *PID1* respectively.
- for *p2*, the XQuery variables $vaildST and $vaildET that are defined within a *LET* clause as an equivalent XQuery expressions for the *p2* expression (**valid**(p2.state/value[**text()**=ST])), as discussed in Sub-section 5.8.2.

5.8.2 The XQuery template for the AIMQL Replay Functions

This sub-section presents the XQuery equivalents for the AIMQL replay functions valid, cast, first, last, overlaps, meets, contains, and precedes.

AIMQL: valid($exp as expression) as variables In AIMQL, the *valid* function returns the time during which the $exp is valid. The XQuery equivalent of the valid function is two XQuery variables, $vaildST and $vaildET, that are defined within a *LET* clause using a specific expression. The variable $vaildST is

assigned the start time of the validity period of the $exp expression , and the variable $vaildET is assigned the end time of the validity period of the $exp expression. The XQuery template for the vaild function is shown in Figure 5.19.

```
let $vaildST :=
db2-fn:xmlcolumn('AIM_ESPlan_TAB.ESPDOC')//{$exp}/xsd:dateTime(@startTime)

let $vaildET :=
db2-fn:xmlcolumn('AIM_ESPlan_TAB.ESPDOC')//{$exp}/xsd:dateTime(@endTime)
```

Figure 5.19: The XQuery template for the *valid($exp as expression)* function.

AIMQL: cast($costnode as node, $unit as String) as xdt: dayTimeDuration In AIMQL, the *cast* function converts the validity period of a specific element to another unit. The support units are second, minute, hour, day, week, month, and year. Figure 5.20 shows the XQuery template for the *cast* function, which is translated into *dayTimeDuration* and any value compared with the cast element or node is translated also into *dayTimeDuration*. Figure 5.21 illustrates the XQuery expression equivalent for the AIMQL expression *cast(p1,hour) <= 10*, such that p1 is an AIMQL variable of type *plan*. In this example the value 10 is translated into *PT10H*, which is a *dayTimeDuration*, and then compared with the validity period of *p1*, which is also translated into *dayTimeDuration*.

```
xdt:dayTimeDuration(
xsd:dateTime($costnode/@endTime) - xsd:dateTime($costnode/@startTime)
)
```

Figure 5.20: The XQuery template for the *cast($costnode as node, $unit as String)* function.

AIMQL: count($exp as as expression) as integer In AIMQL, the *count* function counts how many times the $exp expression is appeared. Figure 5.22 shows the XQuery template for the *count* function, which is translated into the

```
xdt:dayTimeDuration(
xsd:dateTime($p1/@endTime) - xsd:dateTime($p1/@startTime)
) <= xdt:dayTimeDuration("PT10H")
```

Figure 5.21: The XQuery template for the *cast($costnode as node, $unit as String)* function.

corresponding count function in the XQuery, and also $exp is translated into a correct XQuery expression. The function *max, min,* and *avg* are also treated in the same way because these functions are support by the XQuery language.

```
count($exp)
```

Figure 5.22: The XQuery template for the *count($exp as as expression)* function.

AIMQL: first($tNode as temporal node) as node In AIMQL, the *first* function returns the first instance of the temporal node or element. The *first* function is used with the *SHOW* clause. That means the variable $tNode is to be translated into an XQuery variable within *FOR* clause that sets up an iteration over the element associated with the variable $tNode. The XQuery template for *first* is a condition added to the XQuery *WHERE* clause associated with the *FOR* clause and its sub-*FOR* clauses. This condition is that the sub-element's start time should be equal to the initial start time of the $tNode, as shown in Figure 5.23.

```
( xsd:dateTime(@startTime) = xsd:dateTime($tNode/@startTime)
```

Figure 5.23: The XQuery template for the *first($tNode as temporal node)* function.

AIMQL: last($tNode as temporal node) as node In AIMQL, the *last* returns the last or the most recent instance of the temporal node or element. The *last* function is used with the *SHOW* clause. That means the variable $tNode is to be

translated into an XQuery variable within *FOR* clause that sets up an iteration over the element associated with the variable $tNode. The XQuery template for *last* is a condition added to the XQuery *WHERE* clause associated with the *FOR* clause and its sub-*FOR* clause. This condition is that the sub-element's end time should be equal to the end time of the $tNode, as shown in Figure 5.24.

```
( xsd:dateTime(@endTime) = xsd:dateTime($tNode/@endTime)
```

Figure 5.24: The XQuery template for the *first($tNode as temporal node)* function.

AIMQL: overlaps($tNode1 as temporal node, $tNode2 as temporal node) as boolean In AIMQL, the *overlaps* function returns boolean value *true* if the validity period of $tNode1 overlaps the validity period of $tNode2, otherwise it returns boolean value *false*. A validity period *P1* overlaps a validity period *P2*, if the start time of *P1* is less than the end time of *P2* and the start time of *P2* is less than the end time of *P1*. AS shown in Figure 5.25, the XQuery template for *overlaps* is a condition representing the previous semantic added to the XQuery *WHERE* clause associated with the *FOR* clause and its sub-*FOR* clauses. overlaps($tNode1 as temporal node, $tNode2 as validity period) is also supported with the same semantic.

```
( xsd:dateTime($tNode1/@startTime) < xsd:dateTime($tNode2/@endTime) ) and
( xsd:dateTime($tNode2/@startTime) < xsd:dateTime($tNode1/@endTime) )
```

Figure 5.25: The XQuery template for the *overlaps($tNode1 as temporal node, $tNode2 as temporal node)* function.

AIMQL: precedes($tNode1 as temporal node, $tNode2 as temporal node) as boolean In AIMQL, the *precedes* function returns boolean value *true* if the validity period of $tNode1 precedes the validity period of $tNode2, otherwise it

returns boolean value *false*. A validity period *P1* precedes a validity period *P2*, if the end time of *P1* is less than the end time of *P2*. AS shown in Figure 5.26, the XQuery template for *precedes* is a condition representing the previous semantic added to the XQuery *WHERE* clause associated with the *FOR* clause and its sub-*FOR* clause. precedes($tNode1 as temporal node, $tNode2 as validity period) is also supported with the same semantic.

```
( xsd:dateTime($tNode1/@endTime) < xsd:dateTime($tNode2/@endTime) )
```

Figure 5.26: The XQuery template for the *precedes($tNode1 as temporal node, $tNode2 as temporal node)* function.

AIMQL: meets($tNode1 as temporal node, $tNode2 as temporal node) as boolean In AIMQL, the *meets* function returns boolean value *true* if the validity period of $tNode1 meets the validity period of $tNode2, otherwise it returns boolean value *false*. A validity period *P1* meets a validity period *P2*, if the end time of *P1* is equal the end time of *P2*. AS shown in Figure 5.27, The XQuery template for *meets* is a condition representing the previous semantic added to the XQuery *WHERE* clause associated with the *FOR* clause and its sub-*FOR* clause. meets($tNode1 as temporal node, $tNode2 as validity period) is also supported with the same semantic.

```
( xsd:dateTime($tNode1/@endTime) = xsd:dateTime($tNode2/@endTime) )
```

Figure 5.27: The XQuery template for the *precedes($tNode1 as temporal node, $tNode2 as temporal node)* function.

5.8.3 The XQuery Generator

The XQuery Generator parses the AIMQL replay queries and constructs the XQuery equivalent to it. The previous templates are utilized to determine the XQuery equivalent expression for each part of the AIMQL replay query. The generator is aware of the complex information structure and has access to XML Schemas of the skeletal plan (protocol) and the ES plan. The XQuery generator is a module of the *Information Manager* component.

Replay Query Pattern 2:

REPLAY *PLAN* p1
SHOW *When OF FIRST*(p1)
WHERE p1[@domainEntityID = X **and** @protocolID = PID]

The XQuery equivalent to pattern 2:

```
XQUERY
declare namespace xsd ="http://www.w3.org/2001/XMLSchema";
for $p1 in
db2-fn:xmlcolumn('AIM_ESPlan_TAB.ESPDOC')//Plan[@domainEntity_ID=X and @protocol_ID=PID]
where(
( xsd:dateTime($p1/@startTime) = xsd:dateTime($p1/@startTime) )
)
return
<Plan domainEntity_ID="{$p1/@domainEntity_ID}" protocol_ID="{$p1/@protocol_ID}"
startTime="{$p1/@startTime}" endTime="{$p1/@endTime}">
{for $PState in $p1/state
return <state startTime="{$PState/@startTime}" endTime="{$PState/@endTime}">
{ $PState/value[( ( xsd:dateTime($p1/@startTime) <= xsd:dateTime(@endTime) ) and
( xsd:dateTime(@startTime) = xsd:dateTime($p1/@startTime) )
)] }</state> } {for $PSches in $p1/schedules
return <schedules startTime="{$PSches/@startTime}" endTime="{$PSches/@endTime}">
{ for $sch in $PSches/schedule
where (
( xsd:dateTime($sch/@startTime) = xsd:dateTime($p1/@startTime) )
)
return <schedule IDREF="{$sch/@IDREF}" startTime="{$sch/@startTime}" endTime="{$sch/@endTime}">
<scheduleRules startTime="{$sch/scheduleRules/@startTime}" endTime="{$sch/scheduleRules/@endTime}">
{
for $rul in $sch/scheduleRules/rule
where (
( xsd:dateTime($rul/@startTime) = xsd:dateTime($p1/@startTime) )
)
return <rule IDREF="{$rul/@IDREF}" startTime="{$rul/@startTime}" endTime="{$rul/@endTime}">
{for $RState in $rul/state
return <state startTime="{$RState/@startTime}" endTime="{$RState/@endTime}">
{
$RState/value[( ( xsd:dateTime($p1/@startTime) <= xsd:dateTime(@endTime) ) and
( xsd:dateTime(@startTime) = xsd:dateTime($p1/@startTime) )
)]
}</state>
}</rule>
}</scheduleRules></schedule>
}</schedules>
}
</Plan>
```

Figure 5.28: The XQuery script for the AIMQL replay query of pattern 2.

Figure 5.28 provides an the XQuery script for the AIMQL replay query of pattern 2. This XQuery script returns a complete ES plan document that represents the initial plan of the domain entity **X** and this plan is generated from the protocol *PID*.

The XQuery statement in DB2 starts with the key word *XQuery*, as shown in Figure 5.28. It is needed to define the name space used to execute this query, which is the standard W3C XML Schema. This namespace is defined using the key word *declare*. This query pattern has only one variable, *p1*, of type *plan*. Using the variable template, a *FOR* clause is generated to define the XQuery variable **$p1** that iterates over the the plan, whose *domainEntity_ID* attribute is *X* and *protocol_ID* attribute is *PID*. The function *db2-fn:xmlcolumn* is used to access the plans stored in the *AIM_ESPlan_TAB.ESPDOC* table.

The template of the *first* function is used to add the condition that start time of each retrieved element should be equal to the start time of the plan. As shown in Figure 6.5, the XQuery generator is aware of the AIM ESPDoc model. Therefore, the XQuery generator adds sub-FOR clauses, which are required to return the completed initial plan as one XML document.

5.9 Chapter Summary

This chapter has described and discussed the design and implementation of AIMS, the prototype system for managing the complex information. AIMS provides a complete implementation for the AIM language presented in Chapter 4. The main functionalities of the three planes of the SIM framework, which are presented in Chapter 3, are implemented by AIMS. These functionalities are the complex information formalization, instantiation, realization, execution, manipulation and query.

The AIMS system utilizes the modern DBMSs, which provide a triggering mech-

anism and XML storage and retrieval support, to realize the SIM framework and implement the AIM language. In this chapter, the AIMS storage and functionalities are discussed at three levels of abstractions, conceptual, logical and physical. The chapter provides a detailed features that should be provided in the DBMS to be used by AIMS. The AIMS system has been implemented using DB2 and Java.

AIMS developed intermediate models to implement three main components of the AIM language, which are AIMSL (the specification component), AIM ESPDoc (the entity-specific plan model), and AIMQL (the query component).

One of these intermediate models is the *TRME* model, which extends the DBMS triggering mechanism to support the advanced features, such as time-based ECA rules, of the AIMSL rule. Using the *TRME* model, the AIMSL rules are translated into pure SQL triggers managed by the DBMS. The chapter has discussed the limitations of AIMS execution mechanism, which is based on translating the AIMSL rules into triggers, and discussed our solution to these limitations.

The other intermediate model is the *TXME* model that extends the the XML support provided by the modern DBMSs to implement the AIM ESPDoc model. The *TXME* model is consistent and compatible with both XML Schema and the XML data model. Using the *TXME* model, the entity-specific plan documents are stored and retrieved using the modern DBMSs. Based on the *TXME* model, the AIMQL queries are translated into pure XQuery queries, which are executed using the XQuey engine of a modern DBMS.

The chapter has presented our method to calculate the expire date of an entity-specific plan and our method for logging the execution history of the plan. The method used to calculate the expire date is completely implemented using pure SQL statements.

6

Evaluation: A Case Study and Experimental Results

This chapter presents a case study and the experimental results of evaluating the SIM approach and framework supported by the AIM language and the AIMS system. The case study applies the AIM language and the AIMS system to managing a clinical test request protocol. The chapter compares the AIMS systems with another complex information management system, called TOPS. In this chapter, the experiments focus on evaluating the AIMS system specially the AIMS execution mechanism, the storage management for the entity-specific plans, and the execution of AIMQL replay queries.

The chapter is organized as follows: Section 6.1 presents the case study; Section 6.2 compares the AIMS system with the TOPS system; Section 6.3 discusses the experimental results; and Section 6.4 concludes our evaluation to the SIM approach and framework supported by the AIM language and the AIMS.

6.1 Case Study: Applying SIM and AIMS to Managing a Test Request Protocol

This section presents a case study that utilizes the AIMS system to manage a clinical protocol for the diagnosis and treatment of microalbuminuria in diabetes patient. Capturing the knowledge of the microalbuminuria protocol is outside of the case study scope.

Microalbuminuria is diagnosed either on 24 hour urine collections (20 to 200 g/min) or more commonly if elevated concentrations (30 to 300mg/L) on at least two occasions. Albumin levels above these values is called "microalbuminuria", or sometimes just albuminuria. To compensate for the variable possible urine concentration on spot check samples, it is more typical in the UK to compare the amount of albumin in the sample against its concentration of creatinine. This is termed the Albumin/creatinine ratio (ACR) and microalbuminuria is defined as ACR 2.5 mg/mmol (male) or 3.5 mg/mmol(female). The reader is referred to Dube (2004) for more details about the microalbuminuria, which is captured through a research program spanning the Dublin Institute of Technology, Trinity College, and St. James's Hospital.

The case study applies the SIM approach and framework to managing the microalbuminuria protocol (MAP) that incorporated into the activities related to disease management. A experimental and simplified version of the MAP protocol presented in Dube (2004) is formalized and validated using the AIMSL sub-language, and stored in the AIMS XML repository.

Several patient plans (ES plans) are instantiated from the specified MAP protocol (skeletal plan). The execution of these patient plans is managed using the AIMS execution mechanism. The AIM query component, AIMQL, is tested against

both the MAP protocol and patient plans, which represent the complex information produced from incorporating The MAP test request protocol into the diabetes disease management.

6.1.1 The Test Request Protocol Used in the Case Study

An experimental and simplified version of the MAP protocol is specified with focus on covering several cases of events. This experimental version of the MAP protocol contains one schedule, which consists of six rules. Figure 6.1 shows these six rules.

Rule 1 (static Rule, once-off):
event : 2 hours after patient admission
condition: true
action : send a message ordering an ACR test for the patient.

Rule 2 (static rule, repetitive 10 times):
event : every 3 hours after patient admission
condition: true
action : send an observation message.

Rule 3 (Dynamic Rule)
event : When the first result of the ACR test is received
condition: the result value $>$ 35
action : add Rule 4

Rule 4 (static Rule, repetitive 10 times):
event : every week after patient admission
condition: true
action : send an observation message.

Rule 5 (static Rule, repetitive 10 times):
event : every 12 hours after patient admission
condition: the test result $>$ 55
action : send a message ordering an ACR test for the patient.

Rule 6 (static Rule, once-off):
event : 50 hours after patient admission
condition: true
action : remove rule 5

Figure 6.1: The six rules of the experimental version of the MAP protocol utilized in the case study.

Rule 1 orders an *ACR test* for the patient 2 hours after the *patient admission*. *Rule 2* sends an observation message regarding the patient state every 3 hours after the *patient admission* for 10 times. Rule 3 reacts by adding *Rule 4* as soon as the first result of the *ACR test* is received and the result is greater than 35. *Rule*

4 sends an observation message regarding the patient state every week after the *patient admission* for 10 times. Every 12 hours of *patient admission*, *Rule 5* sends an observation message regarding the patient state if the *ACR test* result is greater than 55. *Rule 6* removes *Rule 1* 20 hours after the *patient admission*. It is assumed that the *ACR test* is ordered repeatedly for the patient.

The lifespan of patient plans generated from this protocol ranges from 120 hours to 10 weeks. The duration 120 hours is required if *Rule 4* is not added at the run time. *Rule 4* is to be added if the first result of the *ACR test* is greater than 35. Consequentially, the lifespan of the patient plan is to be 10 weeks.

6.1.2 Applying the SIM Approach and Framework to Patient Plan Management: Dynamic Patient Plan

This section practices the SIM approach and framework in managing the MAP protocol that is incorporated in the activity of disease management. The SIM approach models the MAP protocol as a skeletal plan that could be applied to several patients and adapted to their situations. That means more flexibility in utilizing the MAP protocol and managing the patients.

The SIM framework consists of three plane, the specification, instantiation, and maintenance planes. In the specification plane, a formal specification is generated for the MAP protocol shown in Figure 6.1. The outcome of the specification process is a formal general specification (skeletal plan) for MAP protocol using the AIM specification component, AIMSL.

In the instantiation process, patient plans are instantiated for specific patients form the AIMSL specification of the MAP protocol. These patient plans are realized in the AIMS system by creating their triggers, which represents the reactive logic inherited from the MAP protocol. The instantiated patient plan (*dynamic patient*

plan) contains all the computerised information, about how to react to the changes in the patient conditions. The *dynamic patient plan* is continuously adjusted to the changes in the patient state.

The maintenance plane provides several management aspects for the *dynamic patient plan*. These management aspects are the execution, manipulation, query, and dissemination. In the maintenance plane, the *dynamic patient plan* is:

- dynamically modified and adjusted by its reactive behaviour once one of the interesting clinical events happens
- continuously monitoring the electronic healthcare record to detect the clinical events of interest
- is executed as soon as all its conditions are satisfied

That means the clinicians do not need to continuously monitor the patient state in order to react to the clinical events of interest and adjust the patient plan. The maintenance plane provides the ability to manipulate, query, and disseminate the *dynamic patient plan* and the MAP protocol specification. The clinicians participating in the disease management will be able to remotely access, manipulate or query, the *dynamic patient plan*. Moreover, the *dynamic patient plan* and the MAP protocol specification, which represent the complex information in this application, are subject to traditional and advanced query support, such the replay query support. The task of point-of-care review of a patient plans is made faster and easier by using the replay query support, where the clinicians can review the evolution of a specific patient plan in a particular time period.

6.1.3 The AIMSL Specification for the Test Request Protocol

The formal specification of the utilized MAP protocol is made using the AIMSL sub-language. The outcome of the formal specification process is a well-formed XML document validated against the AIMSL Schema. Figure 6.2 illustrates a browsing view for the MAP protocol specification. The view shows that the MAP protocol has the ID *PRO124*, and belongs to the category, whose ID is *CID124*. As mentioned, it is assumed that each protocol belongs to only one category, and each category contains only one protocol. The protocol consists of five rules, *rule 1*, *2*, *3*, *5*, and *6*. The *rule 4* becomes part of the plan if and only if *rule 3* executed successfully.

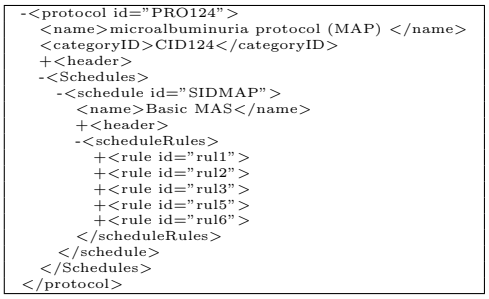

Figure 6.2: the AIMSL specification for the used microalbuminuria protocol (MAP).

Rule 5 is a comprehensive rule that covers several features of the rule element in the AIMSL sub-language. The specification of *rule 5* is illustrated in Figure 6.3, which provides a browsing view focusing on the body of the rule. The rule body consists of the elements (*Terms*, *event*, *condition*, and *action*). There are two terms in *rule 5*. The first term is *value of the ACR test result*, which is a term of type element. Its ID is *TO1234* and its value is of integer data type. The second term is *patient admission*, which is a term of type event. Its ID is *DEPA11*. As discussed in Chapter 4, the term of type element could be used only on the *condition* or *action* element, but the term of type event is used only with the *event* element.

```
-<rule id="rul5">
          <name>Rule 5 of MAP</name>
          +<properties>
          + <header>
          - <body>
            -<Terms >
              <term id="TO1234">
                <title>The value of the ACR test Result</ title >
                <type>element</type>
                <dataType>integer</dataType>
                +<mappingToDB>
              </term>
              <term id="DEPA11">
                <title>patient admission</ title >
                <type>event</type>
                +<mappingToDB>
              </term>
            </Terms >
            -<event id="E1R5">
              <on>
                <relativeTime>
                  <every>
                    <granularity>hours</granularity>
                    <timeLength >12</timeLength>
                    <beforeORafter>
                      <BAValue>after</BAValue>
                      <term id="DEPA11">patient admission</term>
                    </beforeORafter>
                    <for >10</for>
                  </every>
                </relativeTime>
              </on>
            </event>
            -<condition id="ID36">
              +<description>
              <logic>
                <simplePredicate>
                  <operand1>
                    <termID>TO1234</termID>
                  </operand1>
                  <operator>gt</operator>
                  <operand2>
                    <value>
                      <amount>55</amount>
                      <datatype>integer</datatype>
                    </value>
                  </operand2>
                </simplePredicate>
              </logic>
            </condition>
            - <action id="AID36">
              - <do>
                -<proceduralAction>
                  +<sendEMAIL>
                </proceduralAction>
              </do>
            </action>
          </body>
</rule>
```

Figure 6.3: the AIMSL specification for the rule 5.

The *event* element is a repetitive relative time event that happens every 12 (time length) hours (granularity) after the term, whose ID is *DEPA11* that is the *patient admission* term, and the event is repeated 10 times. The *condition* element is a simple predicate checking that the value of the term, whose ID is *TO1234*, is grater than the integer value 55. The action is to send the doctor an email to order an ACR test for the patient.

The specification of *rule 4* is similar to the specification of *rule 5*, except that the granularity of the event is *week* and the *condition* element is true, which means there is no *condition* element. Also, the specification of *rule 2* is similar to the

specification of *rule 5*, except that the time length of the event is 3 and there is no *condition* element. The specification of *rule 1* and *6* are different from *Rule 5* in that their event is *once-off* event. That means the *event* element does not need the *for* element shown in the specification of *rule 5*. The specification of *rule 3* distinguishes in the event type and the action.

6.1.4 A Simulation for the AIMS Execution

This sub-section discusses the execution process of the complex information. The sub-section presents the support provided by the AIMS database schema to simulate as an electronic healthcare record, and discusses the AIMS execution for the generated dynamic patient plans.

6.1.4.1 A Simulated Electronic Healthcare Record

The design of AIMS system does not require to have access to the full electronic healthcare record. The AIMS system has three tables (*Domain_Information*, *Domain_Entity*, and *Category*), in which the information of interest to the MAP protocol is stored.

The *Information Provider* provides the domain information to the AIMS system through messages sent to the AIMS *Communication Manager*. In this case study, the *Information Provider* is the Patient Information System (PIS) that manages the electronic healthcare record. The *Information Provider* (PIS) furnishes the AIMS system with information of interest from PIS electronic healthcare record through messages. The *Information Provider* notifies the AIMS system by the changes of interest.

CATID	CATName	CATDescription
CAT123	Category 1	This category for diabetes renal screening
CAT124	Category 2	This category for general diabetes patient

Table 6.1: the *Category* table.

DEID	CATID	DEName	DEEmail	DEPhone	DEType
PID000	CAT124	Jack O'neil	jo@gmail.com	null	PATIENT
PID001	CAT124	Dan O'neil	do@gmail.com	null	PATIENT
PID002	CAT124	Kevin O'neil	ko@gmail.com	null	PATIENT
.	.	.	.	.	.
PID119	CAT124	Jane O'neil	do@gmail.com	null	DOCTOR
PID002	CAT124	June O'neil	ko@gmail.com	null	DOCTOR

Table 6.2: the *domain entity* table.

The *Category* table has two categories, as shown in Table 6.1. These two categories are one for the general diabetes patient and the other one for the diabetes renal screening. In the case stusy, a simulated patients' contact information is generated and stored in the *Domain_Entity* table, as shown in Table 6.2. The domain entity might be a *patient, doctor*, and *nurse*.

In this simulation, there are 120 domain entities most of them of type patients. Each data item, such as ACR test result and patient temperature, which are used in the skeletal plan (protocol) has a record in the table *Domain_Information* for each patient. For example, if a protocol uses 10 data items and is applied to 10 patients, then the table *Domain_Information* will have 100 records.

DIID	DEID	DIName	DIValue	DIValueNo	DIDataType	DIDescription
TO1234	PID000	ACR Test Result	-99	0	INTEGER	this is an ACR test result
TO1234	PID001	ACR Test Result	-99	0	INTEGER	this is an ACR test result
TO1234	PID002	ACR Test Result	-99	0	INTEGER	this is an ACR test result
TO1234	PID003	ACR Test Result	-99	0	INTEGER	this is an ACR test result
.	.	.	.	.	.	.
TO1234	PID050	ACR Test Result	-99	0	INTEGER	this is an ACR test result
TO1234	PID051	ACR Test Result	-99	0	INTEGER	this is an ACR test result

Table 6.3: the initial *domain information* table.

The MAP protocol uses only one data item, which is the *ACR Test Result* , which is used in rules 3 and 5. AIMS system initializes he table *Domain_Information*, as shown in Table 6.3. Assume, the ID of the *ACR Test Result* is *TO1234*. The initial value for the test is -99, which means that no test result has been received for the patient. Consequentially, the value of *DIValueNo* is zero, which means no test result received. This attribute supports the temporal condition, such as first test result should be greater than 35. The data type of the test result value is integer. Knowing the data type helps to make a correct evaluation for the condition, where

values of same data types are compared with each others.

6.1.4.2 AIMS Execution for the Dynamic Patient Plan

In this case study, 51 patient plans are instantiated from the skeletal plan of the MAP protocol. The patient plans are generated for the patients, whose ID ranges from PID000 to PID050. It is assumed that the patient plan is to be registered 30 minutes after its creation time. Registering a patient plan means creating all its corresponding triggers. After creating the triggers of the patient plan, the plan is in the *active* state waiting to react as soon as a clinical event of interest is detected. That means the *Information Provider* should furnish the AIMS system by the changes in the electronic healthcare record.

```
-<Plan domainEntity_ID="PID050" protocol_ID="PRO124" startTime="2008-01-18T11:53:24" endTime="2999-01-01T01:00:00">
  -<state startTime="2008-01-18T11:53:24" endTime="2999-01-01T01:00:00">
    <value endTime="2008-01-18T12:55:25" startTime="2008-01-18T11:53:24">generated</value>
  </state>
  -<schedules startTime="2008-01-18T11:53:24" endTime="2999-01-01T01:00:00">
    -<schedule IDREF="sch1" startTime="2008-01-18T11:53:24" endTime="2999-01-01T01:00:00">
      -<scheduleRules startTime="2008-01-18T11:53:24" endTime="2999-01-01T01:00:00">
        -<rule IDREF="rul1" startTime="2008-01-18T11:53:24" endTime="2999-01-01T01:00:00">
          <state startTime="2008-01-18T11:53:24" endTime="2999-01-01T01:00:00">
            <value endTime="2008-01-18T12:55:25" startTime="2008-01-18T11:53:24">
              <status>generated</status>
            </value>
          </state>
          +<triggers>
        </rule>
        +<rule IDREF="rul2" startTime="2008-01-18T11:53:24" endTime="2999-01-01T01:00:00">
        +<rule IDREF="rul3" startTime="2008-01-18T11:53:24" endTime="2999-01-01T01:00:00">
        +<rule IDREF="rul5" startTime="2008-01-18T11:53:24" endTime="2999-01-01T01:00:00">
        +<rule IDREF="rul6" startTime="2008-01-18T11:53:24" endTime="2999-01-01T01:00:00">
      </scheduleRules>
    </schedule>
  </schedules>
</Plan>
```

Figure 6.4: The initial patient plan for patient PID050 generated from protocol PRO124.

For simulating the role of the *Information Provider*, a module, which generates a random *ACR test values* ranging from 0 to 100, is developed and attached with the AIMS system. The module generates a value every one hour. Consequentially, although the 51 plans are created from the same skeletal plan, they will be different in their execution and evolution history.

The lifespan of the 51 plans range from 120 hours to 10 weeks, as explained in Sub-section 6.1.1. The lifespan will be 10 weeks only if *rule 4* is added to the

plan. The plans could be classified into two categories, short lifespan (120 hours) and long lifespan (10 week). 18 plans belong to the short lifespan category, and 33 plans belong to the long lifespan category.

Replay Query Pattern 2:

REPLAY *PLAN* p1
SHOW *When OF FIRST*(p1)
WHERE p1[@domainEntityID = "PID050" **and** @protocolID = 'PRO124"]

The XQuery equivalent to pattern 2:

```
XQUERY
declare namespace xsd ="http://www.w3.org/2001/XMLSchema";
for $p1 in
db2-fn:xmlcolumn('AIM_ESPlan_TAB.ESPDOC')//Plan[@domainEntity_ID="PID050" and @protocol_ID='PRO124"]
where(
( xsd:dateTime($p1/@startTime) = xsd:dateTime($p1/@startTime) )
)
return
<Plan domainEntity_ID="{$p1/@domainEntity_ID}" protocol_ID="{$p1/@protocol_ID}"
startTime="{$p1/@startTime}" endTime="{$p1/@endTime}">
{for $PState in $p1/state
return <state startTime="{$PState/@startTime}" endTime="{$PState/@endTime}">
{ $PState/value[( ( xsd:dateTime($p1/@startTime) <= xsd:dateTime(@endTime) ) and
( xsd:dateTime(@startTime) = xsd:dateTime($p1/@startTime) )
)] }</state> } {for $PSches in $p1/schedules
return <schedules startTime="{$PSches/@startTime}" endTime="{$PSches/@endTime}">
{ for $sch in $PSches/schedule
where (
( xsd:dateTime($sch/@startTime) = xsd:dateTime($p1/@startTime) )
)
return <schedule IDREF="{$sch/@IDREF}" startTime="{$sch/@startTime}" endTime="{$sch/@endTime}">
<scheduleRules startTime="{$sch/scheduleRules/@startTime}" endTime="{$sch/scheduleRules/@endTime}">
{
for $rul in $sch/scheduleRules/rule
where (
( xsd:dateTime($rul/@startTime) = xsd:dateTime($p1/@startTime) )
)
return <rule IDREF="{$rul/@IDREF}" startTime="{$rul/@startTime}" endTime="{$rul/@endTime}">
{for $RState in $rul/state
return <state startTime="{$RState/@startTime}" endTime="{$RState/@endTime}">
{
$RState/value[( ( xsd:dateTime($p1/@startTime) <= xsd:dateTime(@endTime) ) and
( xsd:dateTime(@startTime) = xsd:dateTime($p1/@startTime) )
)]
}</state>
}</rule>
}</scheduleRules></schedule>
}</schedules>
}
</Plan>
```

Figure 6.5: the XQuery script for the AIMQL replay query of pattern 2.

The initial patient plan for patient *PID050* generated from protocol *PRO124* is illustrated in Figure 6.4, which provides a browsing view for the initial plan. As shown in the figure, the value *startTime* attribute is *2008-01-18T11:53:24*, which means that the plan was generated on January 18, 2008, at 11:53:24. The value of

the *endTime* attribute is *2999-01-01T01:00:00*, which is used by AIMS to represents the *NOW* value. The AIMS system interprets this value as the current time, at which the query is being processed. The state of the plan is *generated*, and also the state of any sub-element is *generated*. The *value* element of the rule *state* element distinguishes with more details, such as the actual evaluation of its event and condition. These details do not appear because no rule has been executed yet. As shown in Figure 6.4, the initial plan consists of five rules, rule *1, 2, 3, 5,* and *6*.

6.1.5 AIMQL Replay Queries

The AIMQL replay language provides an essential role for retrieving and reviewing the complex information. The user does not need to know the details of the complex information schemas because the AIMQL language is a declarative language. In the following, the AIMQL replay patterns presented in Chapter 4 are used to retrieve and review the progress of the complex information (skeletal plans and ES plans).

The replay pattern number 2 is customized to retrieve the initial patient plan of patient *PID050* generated from protocol *PRO124*, as shown in Figure 6.5. The equivalent XQuery of this AIMQL query is shown in Figure 6.5, and the result of the query is similar to the plan shown in Figure 6.4.

REPLAY *Rule* [@id='rul5'] **R**
SHOW *When OF count*(R.state[value='executed']

Figure 6.6: an AIMQL replay query determining how many times rule 5 is executed.

As discussed, the patient plans, which are created from the same skeletal plan, will be different in their execution and evolution history because of the use of a random value generator. That is evidenced by reviewing how many times *rule 5* is executed. Rule 5 is executed every 12 hours after *patient admission* if the *ACR test result* is greater that *55*. The AIMQL query for determining how many times *rule*

```
XQUERY
for $Plan in db2-fn:xmlcolumn('AIM_ESPlan_TAB.ESPDOC')//Plan
for $R in $Plan//schedule//rule[@IDREF='rul5']
return if ($R/state/value[ status/text()='completed' or status/text()='terminated'])
then
<rule IDREF="rul5" domainEntity="{$Plan/@domainEntity_ID}"
startTime="{$R/@startTime}" endTime="{$R/@endTime}">
<executed>count($R/state/value[ status/text()='executed'])</executed>
</rule>
else
<rule IDREF="rul5" domainEntity="{$Plan/@domainEntity_ID}"
startTime="{$R/@startTime}" endTime="NOW">
<executed>count($R/state/value[ status/text()='executed'])</executed>
</rule>
```

Figure 6.7: The equivalant XQuery script for the AIMQL query determining how many times rule 5 is executed.

5 is executed is shown in Figure 6.6, which defines a variable *R* of type *Rule* with a node test, *[@id='rul5']*, to check that the rule ID is *rul5*, and shows the count of the state executed. The equivalent XQuery script for this AIMQL query is shown in Figure 6.7, which:

- translates the variable *R* of type *Rule* into the XQuery variable *$R* defined in a *FOR* clause.
- adds a new XQuery variable *$Plan* in a FOR clause in order to determines the patient, to who the rule is applied.
- translates the *count(R.state[value='executed'])* into *count($R/state/value[status/text()='executed'])*. As mentioned, AIMS XQuery generator is aware of the Schemas of skeletal plan and ES plan.
- converts the AIMS *NOW* value (*2999-01-01T01:00:00*) to NOW in order to be readable for the user. The semantic of doing that is 1) if the rule is terminated or completed that means the *2999-01-01T01:00:00* value does not exists in as a value for the *endTime* attribute. 2) otherwise, the *2999-01-01T01:00:00* value exists and the NOW value is to replace *2999-01-01T01:00:00*.

Part of the query result is shown in Figure 6.8. For patient (domain entity)

```
<rule IDREF="rul5" domainEntity="PID000" startTime="2008-01-14T12:12:57" endTime="2008-01-19T12:26:07">
<executed>41</executed>
</rule>
<rule IDREF="rul5" domainEntity="PID001" startTime="2008-01-14T12:13:16" endTime="NOW">
<executed>48</executed>
</rule>
<rule IDREF="rul5" domainEntity="PID002" startTime="2008-01-14T12:13:29" endTime="NOW">
<executed>8</executed>
</rule>
<rule IDREF="rul5" domainEntity="PID003" startTime="2008-01-14T12:13:42" endTime="2008-01-19T12:28:10">
<executed>42</executed>
</rule>
<rule IDREF="rul5" domainEntity="PID004" startTime="2008-01-14T12:14:43" endTime="NOW">
<executed>6</executed>
</rule>
<rule IDREF="rul5" domainEntity="PID005"startTime="2008-01-14T12:15:00" endTime="NOW">
<executed>48</executed>
</rule>
<rule IDREF="rul5" domainEntity="PID006" startTime="2008-01-14T12:15:16" endTime="NOW">
<executed>46</executed>
</rule>
<rule IDREF="rul5" domainEntity="PID007" startTime="2008-01-14T12:16:19" endTime="NOW">
<executed>6</executed>
</rule>
<rule IDREF="rul5" domainEntity="PID008" startTime="2008-01-14T12:16:35" endTime="NOW">
<executed>7</executed>
</rule>
.....
.....
.....
<rule IDREF="rul5" domainEntity="PID048" startTime="2008-01-18T11:52:25" endTime="NOW">
<executed>43</executed>
</rule>
<rule IDREF="rul5" domainEntity="PID049" startTime="2008-01-18T11:52:57" endTime="NOW">
<executed>9</executed>
</rule>
<rule IDREF="rul5" domainEntity="PID050" startTime="2008-01-18T11:53:24" endTime="NOW">
<executed>40</executed>
</rule>
```

Figure 6.8: Part of the count query.

number *PID000* and *PID003*, *rule 5* is completed or terminated on 2008-01-19 at 12:26:07 and on 2008-01-19 at 12:28:10, respectively. It is mentioned that in this case study 18 plan are short plans (their lifespan is 120 hours or 5 days) and 33 plan are long plan (their lifespan is 10 weeks).

The patient plan of patients number *PID000* and *PID003* are short plans. the plans after 5 days of the creation time are in the *complete* state; check the difference between the endTime and startTime attributes.

6.2 A Comparison between AIMS and TOPS

This section compares the AIMS system with the TOPS system developed by Dube (2004) in an early stage of this research. The TOPS system is based on an active database management system. This comparison focuses on the complex information storage, temporal rules execution and replay query support. Table 6.4 shows the

results of this comparison.

Criteria	AIMS	TOPS
Complex Information Storage	as one XML document	divided into parts stored into tables
Temporal Rules Execution	managed using the DBMS	managed at the application layer
Replay Queries	supported	not supported

Table 6.4: A comparison between AIMS and TOPS.

6.2.1 Complex Information Storage

TOPS maps the complex information specification into several tables, which represent the TOPS database schema. For the complex information at the generic level, the TOPS database schema consists of 23 tables. For the complex information at the entity-specific level, the TOPS database schema consists of 26 tables. In the TOPS system, the MAP protocol specification used in this case study is to be divided into 23 tables, and an instantiated instance of this specification is to be divided into 26 tables. Consequentially, the complex information retrieval demands join operations, which are a costly operation. Therefore, re-constructing the complex information as one document is a very costly operation in TOPS.

AIMS stores the complex information specification as an XML document. In the AIMS system, the MAP protocol specification and it instantiated instances are to be stored in only one table that has an attribute of XML data type. The complex information retrieval does not demand join operations. Therefore, in the AIMS system there is no need to re-constructing the complex information.

6.2.2 Temporal Rules Execution

TOPS supports the temporal rules execution using a Java based time trigger mechanism implemented at the application layer. This mechanism is used to give signals for the occurrence of the time events that are of interest to the rules of the complex

information. Implementing this mechanism at the application layer means that TOPS is in charge of managing the temporal rules execution. That restricts TOPS to execute only primitive temporal rules. Moreover, this mechanism is restricted to the Java timer capacity.

AIMS temporal rules execution mechanism is based on the *TRME* model that is discussed in Chapter 5. The *TRME* model maps the temporal rules into pure SQL triggers that are completely managed by the DBMS's triggering mechanism. That means all rules of the complex information are managed within the DBMS. AIMS execution mechanism supports advanced temporal rules, which are based on several types of temporal events (relative and absolute) with the ability to be repeated several times.

6.2.3 Replay Queries Support

The TOPS query support is restricted to primitive queries that deal with individual parts of the complex information, such as rules and schedules. The main reason for this restriction is the complicated storage mechanism provided by TOPS for the complex information. To query the complex information as one distinct entity or document, it is a demand to join more than 20 tables. Therefore, TOPS did not support the replay queries over the complex information.

AIMS provides a replay query support that plays over again the history of the complex information to show the in details the actions that cause changes during the complex information life span. This replay query support deals with the complex information as a whole or its individual parts. Moreover, the AIMS replay support is able to deal with several complex information instances, such as replay several patient plans according to specific condition. The AIMS system maps the replay queries into pure XQuery queries that are executed using the utilized DBMS.

6.3 Experimental Results

This section discusses the experimental results that focus on evaluating the AIMS system specially the AIMS execution mechanism, the storage management for the entity-specific plans, and the execution performance of AIMQL replay queries. These experiments are tested on Debian 4, a Linux system, and an Intel Pentium III processor machine, whose configuration is one Gigabyte RAM and 40 Gigabyte hard disk.

6.3.1 The Experimental Results of the AIMS Execution Mechanism

The objective of this experiment is to demonstrate the performance of AIMS execution mechanism using the time spent for updating the *timing event table* as performance metric. This time includes the time required to process all the triggers fired at the updating time. The minimum granularity supported for the rules used in this experiment is an hour, and the repetition period of the job scheduler is 30 minutes.

The focus of the experimental results is on the memory and the job schedule task time. The AIMS execution mechanism utilizes the DBMS job scheduler (task centre) to periodically update the *timing event table* of each plan. Once the plan timing table is updated the triggers are fired and the its conditions are to be evaluated.

Figure 6.9 illustrates the system performance with regard to the number of concurrent triggers using the average elapsed time of executing the job scheduler (task), which is calculated by the DB2 task centre in DB2. Our empirical results demonstrate that the performance of our system is exponential in the number of triggers fired by the system at the update time.

The current system performance is to be improved through system's resources

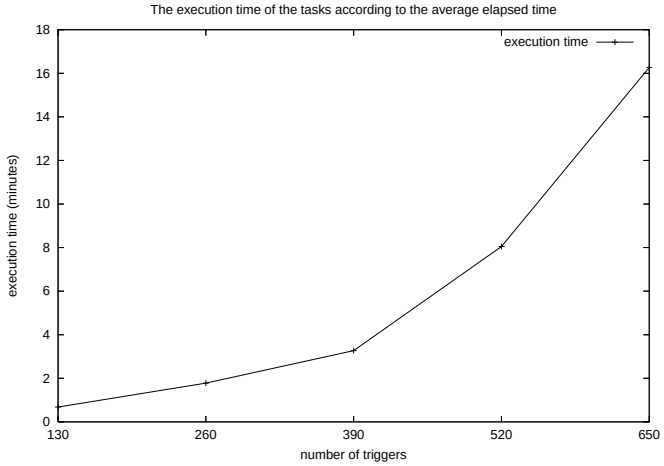

Figure 6.9: The execution time according to the average elapsed time.

optimization techniques. The main performance factors of the AIMS execution mechanism are:

(1) The number of concurrent triggers, which are invoked at the same time. The size of the heap used to managing the concurrent triggers determines the performance of executing concurrent triggers.

(2) The size of the plan. The plan is growing over time. That affects the time required to log the plan execution history. Consequentially, the elapsed time time of the task is affected.

6.3.2 The Experimental Results of the ES Plan Document Size

The objective of this experiment is to demonstrate the AIMS storage management performance. AIMS manages the complex information (skeletal plans and ES plans) and the domain information. The complex information is stored as XML documents. Both the skeletal plans and domain information, which is stored in relational tables, are non-temporal data. The ES plan is a temporal XML document that records all the changes produced by updating the ES plan. Most of these changes add a new state to an element of the ES plan. For example, executing

Rule 2 every three hours adds a new *executed* state under the *rule* element. These changes might be also adding a new rule, such as *Rule 3* might add a new rule, *Rule 4*. Consequentially, The storage management of the ES plans is of critical importance and the main factor of the AIMS storage management performance.

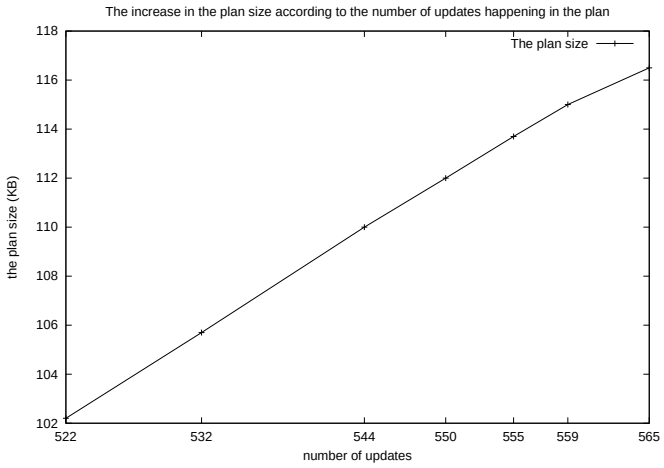

Figure 6.10: The correlation between the ES plan growing size and the number of updates happening in the plan.

This experiment compares the size of ES plans with the number of updates that take place in them. The growing in the plan size is almost linear to the number of updates, as shown in Figure 6.10. The linear relationship between the ES plan size and number of updates assists in estimating the ES plan size after N number of updates, such that most of the updates are changes on the rule state. The AIMS storage management is stable to the number of updates.

This linear graph shown in Figure 6.10 aids in illustrating a two dimensional relationship (equation) between the ES plan size (Y) and the number of updates (X), where 1) the slope of the line is *0.342* and 2) the y-intercept, which gives the point of intersection between the graph of the function and the y-axis, is *-76.27*. This information represents the equation between the ES plan size (Y) and the number of updates (X), as the following: *Y= 0342X - 76.27*. Using this equation, the storage required for managing a specific skeletal plan with N ES plans is to be

estimated.

6.3.3 The Experimental Results of the AIMQL Replay Queries

The objective of this experiment is to demonstrate the AIMS query performance. AIMS translates the AIMQL replay queries into a pure XQuery, which is executed by the DB2 XQuery engine. DB2 provides different tools, such as *db2batch*, to analyse the runtime performance of queries. The *db2batch* returns the elapsed time spent for executing the given query. The ES plans is of critical importance and the main factor of the AIMS query performance because the ES plan documents grow over time.

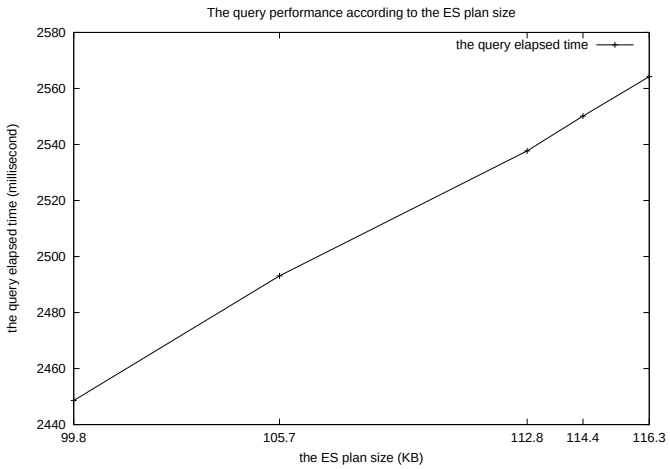

Figure 6.11: The The correlation between the query execution time and the size of the ES plan.

This experiment compares the query execution elapsed time with the size of ES plans, which accessed in the query. The experiment is achieved using a complicated query, which accesses an ES plan and scans it three time for calculating the number of executing its rules and returning the recent instance of the plan. The long lifespan plans are utilized in this experiment. The query runtime performance is almost linear to the size of the ES plan participating in the query, as shown in Figure 6.11.

The equation, *Y= 6.9 X + 1760*, is formalized from the the linear graph shown in Figure 6.11, which illustrates a two dimensional relationship between the query execution elapsed time (Y) and the ES plan size (X). The slope of the line is *6.9*. The y-intercept is *1760*. Using this equation, the AIMQL replay queries is to be estimated.

6.4 Concluding Remarks

The AIMS system with the AIM language and the SIM approach and framework are evaluated with respect to the following software quality attributes: maintainability, extensibility, reusability, and performance.

6.4.1 Maintainability

The SIM approach uses a declarative language, AIM, to allow a unified management to the domain knowledge. The AIM language formalizes the domain knowledge as skeletal plans at the level of what to do, not how to do it, thus making it easy to incorporate and maintain the domain knowledge into application activities. The AIM language facilitates the creation and maintenance of the entity-specific plans generated from a specific skeletal plan. As shown in the case study, several patient plans, which are created from the MAP protocol, are to be easily edited and redeployed. Furthermore, The AIM language allows testing and validating the changes to the skeletal plans and the ES plans immediately using the AIM XML Schemas.

6.4.2 Extensibility

Extending the domain knowledge or specific skeletal plans can be deployed easily using the AIM manipulation operations. That means new skeletal plans, which

represents the domain knowledge required for a specific activity, can be easily added to the existing AIMSL specification stored in the AIMS repository. Adding skeletal plans is easily deployed because it does not required changes in the system, such as in workflow systems changing the utilized the domain knowledge means changing the workflow system. The skeletal plans are deployed through generating ES plans, which are realized in the system by registering its triggers.

6.4.3 Reusability

The domain knowledge is specified as interpretable format using AIMSL. The similar application could reuse this AIMSL specification. In the same application domains, the domain knowledge is almost similar. Thus, the AIMSL specification could be reused. Also, the SIM framework provides the customization process to be used to adapt the skeletal plans (AIMSL specification) to the organization needs.

6.4.4 Performance

Performance is the main software attribute in evaluating AIMS system. The AIMS system utilizes the modern DBMSs, and AIMS execution mechanism are based on the DBMS triggering information retrieval mechanisms. Therefore, AIMS performance is correlated with the utilized DBMS. The AIMS execution performance is exponential to the concurrent triggers, which could be reduced by providing time-based optimization. In order to reduce the number of concurrent triggers, the time-based optimization focuses on detecting in advance the triggers that should not be fired based on the triggers time-based events, which are expressed as predicates in the triggers *when* clause. The AIMS storage performance is linear to the number of updates taking place in the ES plans. A linear equation is to be used to estimate the required storage for executing ES plans of a specific skeletal plan.

The AIMS query performance is also linear to the ES plans size.

Now is not the end.

It is not the beginning of the end.

It is perhaps, the end of the beginning.

Winston Churchill

7

Conclusion

This chapter briefly review this thesis, summarises the thesis contributions, then presents the future work related to the concepts developed in this thesis.

7.1 Thesis Review

This thesis has investigated the modelling of the complex information and its management in order to support the day-to-day organization activities. The complex information consists of two main parts; *active* and *passive*. The *active* part determines the recommended procedure that should be taken as a react to specific situations. The *passive* part determines the information that describes these situations and other descriptive information plus the execution history of the complex information. In the healthcare domain, the patient plan is an example for complex information produced during the disease management from specific clinical guidelines. For this investigation, the main research questions defined to be answered

within this thesis are:

- What is a suitable way to model and manage the complex information produced during the day-to-day organization activities that apply domain knowledge, such as clinical guidelines?
- How to facilitate and realize the model of the complex information and the management aspects using a unified language?
- How to utilize the modern DBMS, which support XML technologies and triggering mechanism, to realize this language?

This thesis has started by analysing the different ways or approaches proposed for modelling and managing the complex information. The most related approaches are proposed in the area of workflow management and the computerised clinical guidelines. The first part of Chapter 2 aimed at justifying the shortcomings of these approaches and setting a clear distinction between managing the *active* part of the complex information and the complex information itself. The second part of Chapter 2 aimed at analysing the XML-based ECA rule languages using a comparative framework, called CoAX. The main criteria of the CoAX framework specified according to the needs of the complex information management. The main findings of Chapter 2 are the need to 1) an approach and framework for managing the complex information at a domain and high level; and 2) an advanced language overcomes the shortcomings of the XML-based ECA rule language.

This thesis has presented in Chapter 3 the SIM approach and framework for managing the complex information. Figure 7.1 shows the SIM approach and framework. The SIM approach provides a conceptual model for the complex information. This model design the complex information as skeletal plans from which several entity-specific plans are generated. The skeletal plans and its corresponding entity-

specific plans represent the complex information produced from incorporating domain knowledge into organization activities.

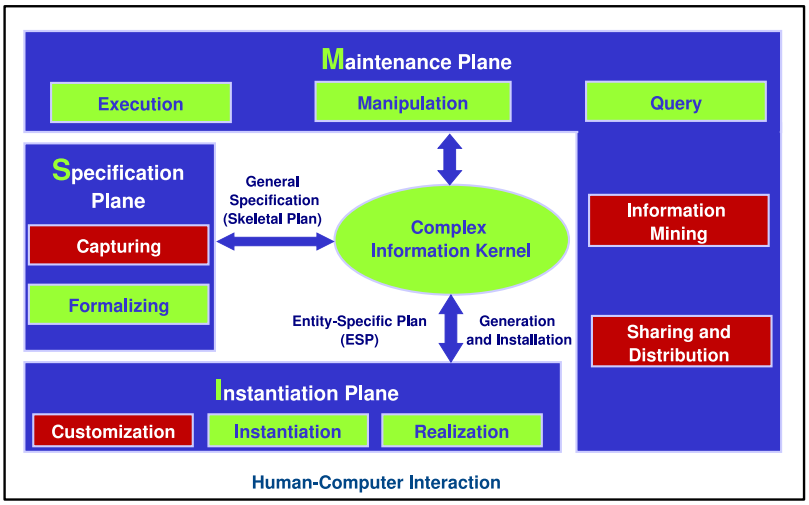

Figure 7.1: SIM: A generic approach and framework for computerising the Complex Information.

The SIM framework provides comprehensive management aspects for managing the complex information. In the SIM framework, the complex information goes through three phases, specifying the skeletal plans, instantiating entity-specific plans, and then maintain these entity-specific plans during their lifespan. Consequently, these management aspects are classified into three planes, *specification*, *instantiation*, and *maintenance*. The specification plane includes the *capturing* and *formalization* aspects. The instantiation plane includes the *customisation*, *instantiation*, and *realization* aspects. The maintenance planes includes the *execution*, *manipulation*, *query*, *information mining*, and *sharing and distribution* aspects. The base of the three planes is a human-computer interaction support, as shown in Figure 7.1.

Chapter 4 has presented the AIM language, which is developed to support the SIM approach and framework. It is a high-level, declarative, and XML-based language that is divided into three components, AIMSL, AIM ESPDoc model, and

AIMQL. The AIMSL is the AIM specification component that support the formalization process of the best practice as skeletal plans that is represented as XML document. The AIMSL model is based on the ECA rule paradigm with extensions to support temporal events and conditions at the application domain level. The AIM ESPDoc model is a physical model for the entity-specific plan. This model represents the entity-specific plan as a temporal XML document, which is at the same time well-formed XML document. The AIM language specifies the complex information; the skeletal plans and entity-specific plans as XML document that is to be stored in any XML database. The third component is the AIMQL, which is the AIM query component. AIMQL provides support for manipulating and querying the complex information, and provides special manipulation operations and query capabilities for the entity-specific.

Chapter 5 has presented the AIMS system, which utilizes the available database management systems (DBMS) as a base for managing the complex information and implementing the AIM language. AIMS developed two intermediate models. One of these intermediate models is the *TRME* model, which extends the DBMS triggering mechanism to support the advanced features, such as time-based ECA rules, of the AIMSL rule. Using the *TRME* model, the AIMSL rules are translated into pure SQL triggers managed by the DBMS. The other intermediate model is the *TXME* model that extends the XML support provided by the modern DBMSs to implement the AIM ESPDoc model. The *TXME* model is consistent and compatible with both XML Schema and the XML data model. Using the *TXME* model, the entity-specific plan documents are stored and retrieved using the modern DBMSs. Based on the *TXME* model, the AIMQL queries are translated into pure XQuery queries, which are executed using the XQuery engine of a modern DBMS.

Chapter 6 has discussed our evaluation to the SIM approach and framework supported by the AIM language and the AIMS system. Our case study has applied the AIM language and the AIMS system to managing a test request protocol. Our experiments focus on evaluating the AIMS system specially the AIMS execution mechanism based on the *TRME* model, the AIMS repository based on the *TXME* model, and the AIMS queries performance. These experiments are tested on Debian 4, a Linux system, and an Intel Pentium III processor machine, whose configuration is one Gigabyte RAM and 40 Gigabyte hard disk. The experimental results show that:

- the AIMS repository utilizes the storage in an efficient way, where the growing in the entity-specific (ES) plan size is linear to the number of updates;
- the AIMS query performance is linear to the size of the ES plan participating in the query;
- the performance of the AIMS execution mechanism is exponential in the number of concurrent triggers. This performance is to be enhanced using resource optimization techniques to increase the capacity of the used machine and time-based optimization to reduce the number of concurrent triggers.

7.2 Summary of Thesis Contributions

This thesis contributions are summarised as follows:

- A discussion of the shortcomings of approaches addressing the complex information management, and the identification of a need for an empirical approach to managing the complex information at an application domain and end-user level.

- A comparative framework, called CoAX, for analysing the available XML-based ECA Rule languages. The CoAX framework considered the requirements demanded to support the complex information management, and aims at determining shortcomings of these languages.

- The development of the SIM approach for modelling the complex information as one distinct entity, which consists of two main parts; *active* and *passive*. The *active* part determines the recommended procedure that should be taken in specific situations. The *passive* part determines the information that describes these situations and other descriptive information plus the execution history of the complex information.

- The development of the SIM framework for managing the complex information through three planes; *specification*, *instantiation*, and *maintenance*. The SIM framework is a generalized and enhanced version of the SpEM framework developed in an early stage of this research by (Dube 2004).

- The development of the AIM language that facilitates the main management aspects of the SIM framework, and provides a computer-interpretable model for the complex information according to the SIM approach. The AIM language consists of three components, *AIMSL* for specifying the complex information, *AIM ESPDoc* for modelling the complex information instances and *AIMQL* for manipulating and querying the complex information. *AIMSL* extends the functionality of *PLAN* specification language and enriches the rule paradigm of *PLAN*, which was developed in an early stage of this research by (Wu and Dube 2001).

- An implementation of a proof-of-concept systems, called AIMS, to demonstrate that the method developed in this thesis can be applied in practice. AIMS

develops two intermediates models: a model called *TRME* for extending the available DBMS triggering mechanism to support temporal rules defined at a domain and high level; another model called *TXME* for extending the XML database to support temporal data.

- An evaluation to the AIMS system through a clinical case study applied to a test request protocol. The evaluation focuses on appraising the AIMS execution mechanism, storage technique and query performance. The overall evaluation shows a good support to the test request application.

7.3 Future Work

Several management aspects of the SIM framework shown in Figure 7.1 were out of the scope of this thesis. These management aspects are *capturing*, *customisation*, *information mining*, *sharing and distribution* and the *human-computer interaction* support. These management aspects poses major challenges for data mining techniques, distributed and mobile information management, and natural language processing. The main projects required to cover these management aspects and an extension to the AIM language are summarised below.

7.3.1 AIMQL Visualisation Mechanism

The AIMQL replay queries return a temporal XML document, which represents the replay of the complex information execution. This replayed information is visualised as a text that could be browsed using any XML or Web browser. This visualisation mechanism is very simple and does not provide a domain and high level view to the replayed information. It is needed to develop an advanced graphical visualisation mechanism to review the replayed information in a way similar to a movie. This visualisation mechanism should consider the semantic of the complex information

and provides functionalities similar to the functionalities provided by a movie player.

7.3.2 The Information Mining

The information mining project is to develop a method that provides automatic discovery of information from an evolution history component of the entity-specific plan, which represents a real case study. This discovered information can be used to deploy new best practices or as a feedback tool that helps in auditing, analysing and improving already enacted best practices.

7.3.3 The Distributed and Mobile Management

The distributed and mobile management is to investigate into supporting the distributed execution, manipulation, and query, and provide a mobile information system for the complex information management. The distributed manipulation and query should overcome the heterogeneity fragmentation of the information. The distributed execution requires distributed event detection, condition evaluation and action. The time difference between geographically dispersed organization and users should be taken into account in executing time-based rules. Using the mobile devices, such as Personal Digital Assistant (PDA), as a client for AIMS system facilitates the nature of the modern organization activities, where the users or stockholders demand a remote access and management for the complex information.

7.3.4 The Human-Computer Interaction support

The Human-Computer Interaction support is to investigate into providing nature language support for the three planes of the framework. It is difficult to the end users to understand and review the skeletal plans and the entity-specific plans at the low level. The nature of the best practice and its complex information as a huge

amount of advanced information should be considered as an essential factor for the user interface in two directions. The first direction is to translate from a natural language, in context of best practices, into a formal specification that the system can process further. The second direction is to translate the complex information from a physical and low level representation into a human readable and high level representation model.

References

Abiteboul, S., Benjelloun, O., Manolescu, I., Milo, T., and Weber, R., 2002. Active XML: A data-centric perspective on Web services. In *BDA*.

Abraham, T. and Roddick, J., 1999. Survey of spatio-temporal databases. *Geoinformatica*, **3**(1):61–99. article.

Al-Kateb, M., E., E.-S. M., and Osman, H., 2003. Towards event mining: Representing real-world events in temporal databases. In *Al-Azhar Engineering 7th International Conference*, Cairo. inproceedings.

Andrew, E. and Melton, J., 2002. SQL/XML is making good progress. *SIGMOD Rec.*, **31**(2):101–108. article.

Arabshian, K. and Schulzrinne, H., 2003. A SIP-based medical event monitoring system. *5th International Workshop on Enterprise Networking and Computing in Healthcare Industry, Healthcom*, :66– 70. Santa Monica, CA.

Arciniegas, F., 2000. *XML Developer's Guide*. McGraw-Hill Companies. book.

Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J., 2002. Models and issues in data stream systems. In *PODS*, pages 1–16.

Babu, S. and Widom, J., 2001. Continuous queries over data streams. *SIGMOD Rec.*, **30**(3):109–120. article.

Bailey, J., Bry, F., Eckert, M., and Patranjan, P.-L., 2005. Flavours of XChange, a rule-based reactive language for the (semantic) Web. In *RuleML*, pages 187–192.

Bailey, J., Poulovassilis, A., and Wood, P. T., 2002a. Analysis and optimisation of event-condition-action rules on XML. *Computer Networks*, **39**(3):239–259. article.

Bailey, J., Poulovassilis, A., and Wood, P. T., 2002b. An event-condition-action language for XML. In *The 12th International World Wide Web Conference, www*, pages 486–495, Hawaii. inproceedings.

Berglund, A., Boag, S., Chamberlin, D., Fernndez, M. F., Kay, M., Robie, J., and Simon, J., 2005. XML path language (XPath) 2.0. Technical report, W3C Working Draft. techreport.

Bernauer, M., Gerti, K., and Gerhard, K., 2004. Composite events for XML. In *the 13th International World Wide Web Conference, WWW13*, pages 175–183, New York, U.S.A. inproceedings.

Boag, S., Chamberlin, D., Fernandez, M., Florescu, D., Robie, J., and Simeon, J., 2007. XQuery 1.0: An XML query language. Technical report, W3C Recommendation. techreport.

Bonifati, A., 2000. Active behaviors within XML document management. In *WorkShop EDBT Ph.D.*, Konstanz (Germany). EDBT Ph.D. WorkShop. inproceedings.

Bonifati, A., 2001. *Reactive Services for XML Repositories*. PhD thesis, Politecnico di Milano. phdthesis.

Bonifati, A., Braga, D., Campi, A., and Ceri, S., 2002. Active XQuery. In *ICDE*, pages 403–.

Bray, T., Paoli, J., and Sperberg-McQueen, C. M., 1998. Extensible markup language (XML) 1.0. Technical report, W3C Recommendation.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F., 2008. Extensible markup language (XML) 1.0 (fifth edition). Technical report, W3C Proposed Edited Recommendation.

Bry, F., Eckert, M., Patranjan, P.-L., and Romanenko, I., 2006. Realizing business processes with ECA rules: Benefits, challenges, limits. In *Principles and Practice of Semantic Web Reasoning (PPSWR)*, pages 48–62.

Caironi, P., Portoni, L., Combi, C., Pinciroli, F., and Ceri, S., 1997. HyperCare: a prototype of an active database for compliance with essential hypertension therapy guidelines. In Masys, D. R., editor, *AMIA Ann Fall Symposium*, pages 288–292, Philadelphia, PA. Hanley and Belfus. inproceedings.

Casteleiro, M. A. and Diz, J. J. D., 2008. Clinical practice guidelines: A case study of combining owl-s, owl, and swrl. *Knowledge-Based Systems*, **21**(3):247–255.

Cerami, E., 2002. *Web Services Essentials*. O'Reilly. book.

Ceri, S., Cochrane, R., and Widom, J., 2000. Practical applications of triggers and constraints: Success and lingering issues (10-year award). In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 254–262. Morgan Kaufmann Publishers Inc. inproceedings.

Chamberlin, D., Florescu, D., Robie, J., Simeon, J., and Stefanescu, M., 2001. XQuery: A Query language for XML. Technical report, W3C Working Draft.

Chandra, R. and Arie, S., 1994. Active databases for financial applications. In *Fourth IEEE Research Issues in Data Engineering: Active Database Systems (RIDE-ADS)*, pages 46–52. inproceedings.

Chen, W.-J., Sammartino, A., Goutev, D., Hendricks, F., Komi, I., Wei, M.-P., and Ahuja, R., 2006. *DB2 9 pureXML Guide*. IBM Redbooks, first edition edition.

Cho, E., Park, I., Hyum, S. J., and Kim, M., 2002. ARML: an active rule mark-up language for heterogeneous active information systems. In *Proceedings of the*

International Workshop on Rule Markup Languages for Business Rules on the Semantic Web, Sardinia, Italy. inproceedings.

Ciccarese, P., Caffi, E., Boiocchi, L., Halevy, A., Quaglini, S., Kumar, A., and Stefanelli, M., 2003. The NewGuide project: guidelines, information sharing and learning from exceptions. *the 9th Conference on Artificial Intelligence, in Medicine in Europe, AIME*, :163–167.

Clayton, P., Pryor, A., Wigertz, O., and Hripcsak, G., 1989. Issues and structures for sharing knowledge among decision-making systems: The 1989 arden homestead retreat. In *Proceedings of the Thirteenth Annual Symposium on Computer Applications in Medical Care*, pages 116–121. IEEE Computer Society Press. inproceedings.

Clercq, P. D., Blom, J. A., Korsten, H. H. M., and Hasman, A., 2004. Approaches for creating computer-interpretable guidelines that facilitate decision support. *Artificial Intelligence in Medicine*, **31**(1):1–27.

Curran, T. A., Keller, G., and Ladd, A., 1997. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Enterprise Resource Planning Series.

Dart, T., Yigang, X., Gilles, C., and Patrice, D., 2001. Computerization of guidelines: Towards a "guideline markup language". *Medinfo*, **10**:186–190. article.

DeHaan, D., David, T., P., C. M., and Tamer, O. M., 2003. A comprehensive XQuery to SQL translation using dynamic interval encoding. In *SIGMOD*, San Diego, CA. inproceedings.

Dube, K., 2004. *A Generic Approach to Supporting the Management of Comput-*

erised Clinical Guidelines and Protocols. PhD Thesis. PhD thesis, Dublin Institute of Technology (DIT). phdthesis.

Dube, K., Wu, B., and Grimson, J., 2002. Using ECA rules in database systems to support clinical protocols. In *13th International Conference on Database and Expert Systems (DEXA 2002)*, pages 226–235. inproceedings DBLP:conf/dexa/2002.

Elmasri, R. and Navathe, S., 2003. *Fundamentals of Database Systems*. Addison Wesley, 4 edition.

Fallside, D. C. and Priscilla, W., 2004. XML schema part 0: Primer second edition. Technical report, W3C Recommendation. techreport.

FAO, 2003. Development of a framework for good agricultural practices. Technical report, Food and Agriculture Organization of the United Nation (F A O), http://www.fao.org/docrep/meeting/006/y8704e.htm.

Field, M. J. and Lohr, K. N., 1992. *Guidelines for Clinical Practice: From Development to Use*. National Academy Press, Washington, DC. book.

Florescu, D. and Donald, K., 1999. Storing and querying XML data using an RDMBS. *IEEE Data Eng. Bull.*, **22**(3):27–34. article.

Georg, G. and Jaulent, M.-C., 2007. A document engineering environment for clinical guidelines. In *DocEng '07: Proceedings of the 2007 ACM symposium on Document engineering*, pages 69–78, New York, NY, USA. ACM.

Golab, L. and Tamer, O. M., 2003. Issues in data stream management. *SIGMOD Rec.*, **32**(2):5–14. article.

Goralwalla, I. A., U., T. A., and Tamer, O. M., 1995. Experimenting with temporal relational databases. In *CIKM '95: Proceedings of the fourth international con-*

ference on Information and knowledge management, pages 296–303, New York, NY, USA. ACM Press. inproceedings.

Gottschalk, F., van der Aalst, W. M. P., and Jansen-Vullers, M. H., 2007. Sap webflow made configurable: Unifying workflow templates into a configurable model. In *Business Process Management*, pages 262–270.

Graham, S., Niblett, P., Chappell, D., Lewis, A., Nagaratnam, N., Parikh, J., Patil, S., Samdarshi, S., Tuecke, S., Vambenepe, W., *et al.*, 2004. Web services notification (ws-notification). Technical report, The University of Chicago.

Greenes, R. A., Peleg, M., Boxwala, A., Tu, S., Patel, V., and Shortliffe, E., 2001. Sharable computer-based clinical practice guidelines: Rationale, obstacles, approaches, and prospects. In *Medinfo 2001*, pages 201–5, London, UK. inproceedings.

Grimshaw, J. M. and Russell, I., 1993. Effect of clinical guidelines on medical practice: a systematic review of rigorous evaluations. *Lancet*, **342**:1317–22.

Grimson, W., Berry, D., Grimson, J., Stephens, G., Felton, E., Given, P., and O'Moore, R., 1998. Federated healthcare record server - the synapses paradigm. *International Journal of Medical Informatics*, **52**:3–27.

Grust, T., Sakr, S., and Teubner, J., 2004. XQuery on SQL hosts. In *Proceedings of the 30th VLDB Conference*, Toronto, Canada. inproceedings.

Halamka, J. D., Osterland, C., and Safran, C., 1998. CareWeb TM, a Web-based medical record for an integrated healthcare delivery system. *Medinfo 98*, . article.

He, H., Haas, H., and Orchard, D., 2004. Web services architecture usage scenarios. Technical report, W3C Working Group Note. techreport.

Hors, A. L., Hogaret, P. L., Wood, L., Nicol, G., Robie, J., Champion, M., and Byrne, S., 2000. Document object model (DOM) level 2 core specification. Technical report, W3C Recommendation. techreport.

Hripczak, G., Clayton, P. D., Jenders, R. A., Cimino, J. J., and Johnson, S. B., 1996. Design of a clinical event monitor. *Comput. Biomed. Res.*, **29**(3):194–221.

Janssens, G. K., Verelst, J., and Weyn, B., 2000. Techniques for modeling workflows and their support of reuse. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 1–15, London, UK. Springer-Verlag.

Jenders, R. A., Huang, H., Hripcsak, G., and Clayton, P. D., 1998. Evolution of a knowledge base for a clinical decision support system encoded in the arden syntax. In *AMIA Symp*, pages 558–562. inproceedings.

Jones, B., Abidi, S. S. R., and Ying, W., 2005. Using computerized clinical practice guidelines to generate tailored patient education materials. In *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05)*, page 139.2, Washington, DC, USA. IEEE Computer Society.

Kappel, G., Rausch-Schott, S., and Retschitzegger, W., 1998. Coordination in workflow management systems - a rule-based approach. In *Coordination Technology for Collaborative Applications - Organizations, Processes, and Agents [ASIAN 1996 Workshop]*, pages 99–120, London, UK. Springer-Verlag.

Kappel, G., Rausch-Schott, S., and Retschitzegger, W., 2000. A framework for workflow management systems based on objects, rules and roles. *ACM Comput. Surv.*, **32**:27.

Keller, G. and Teufel, T., 1998. *SAP R/3 Process Oriented Implementation: Iterative Process Prototyping*. Addison-Wesley Professional.

Khanda, A., Gemmellb, I., Rankina, A., and Cleland, J., 2000. Clinical events leading to the progression of heart failure: insights from a national database of hospital discharges. *European Heart Journal*, **22**(2):153–164.

Kiepuszewski, B., ter Hofstede, A. H. M., and van der Aals, W. M. P., 2003. Fundamentals of control flow in workflows. *Acta Informatica*, **39**(3):143–209.

Kiyomitsu, H., Atsunori, T., and Katsumi, T., 2001. Activeweb: XML-based active rules for Web view derivations and access control. :31–39. article.

Knolmayer, G., Endl, R., and Pfahrer, M., 2000a. Modeling processes and workflows by business rules. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 16–29, London, UK. Springer-Verlag.

Knolmayer, G., Endl, R., and Pfahrer, M., 2000b. Modeling processes and workflows by business rules. *Business Process Management*, **1806/2000**:201–245.

Kulkarni, K. G., Nelson, M., and Roberta, C., 1999. Active database features in SQL3. In *Active Rules in Database Systems*, pages 197–219. incollection.

Lu, R. and Sadiq, S., 2007. A survey of comparative business process modeling approaches. In *Business Information Systems (BIS)*, volume 4439/2007, pages 82–94. Springer Berlin / Heidelberg.

Manola, F. and Eric, M., 2004. RDF primer. Technical report, W3C Recommendation. techreport.

Mansour, E., 2003. *Applying Temporal Database Storage and Retrieval Techniques to XML Data*. PhD thesis, the Faculty of Computers and Information Cairo University. mastersthesis.

Mansour, E., Dube, K., and Wu, B., 2007. AIM: An XML-based temporal and ECA rule language for managing complex information. In *International RuleML Symposium on Rule Interchange and Applications (RuleML 2007)*, Orlando, Florida.

Mark Scardina, B. C. J. W., 2004. *Oracle Database 10g XML & SQL: Design, Build, & Manage XML Applications in Java, C, C++, & PL/SQL*. McGraw-Hill Osborne Media. book emansour.

McCarthy, D. and Umeshwar, D., 1989. The architecture of an active database management system. In *ACM SIGMOD international conference on Management of data*, pages 215–224, New York, NY, USA. ACM Press. article.

McDonald, C., 1976. Use of a computer to detect and respond to clinical events: its effect on clinician behavior. *Ann Intern Med.*, **84**(2):162–167.

Mller, A. and Schwartzbach, M. I., 2006. *An Introduction to XML and Web Technologies*. Addison Wesley.

Müller, D., Reichert, M., and Herbst, J., 2006. Flexibility of data-driven process structures. *Business Process Management Workshops*, **4103/2006**:181–192.

Müller, R., 2002. *Event-Oriented Dynamic Adaptation of Workflows: Model, Architecture and Implementation*. PhD thesis, Department of Computer Science, University of Leipzig.

Müller, R., Greiner, U., and Rahm, E., 2004. $\mathrm{A_{gent}W_{ork}}$: a workflow system supporting rule-based workflow adaptation. *Data & Knowledge Engineering*, **51**(2):223–256.

Nicola, M. and Linden, B. V. d., 2005. Native XML support in DB2 universal database. In *VLDB*, pages 1164–1174, Trondheim, Norway.

Orriëns, B., Yang, J., and Papazoglou, M. P., 2003. A framework for business rule driven service composition. In *Technologies for E-Services (TES)*, pages 14–27.

Papamarkos, G., Alexandra, P., and T., W. P., 2003. Event-condition-action rule languages for the semantic Web. In *Workshop on Semantic Web and Databases (SWDB), at VLDB'03*, pages 309–327. inproceedings DBLP:conf/semweb/2003swdb.

Paton, N., 1999. *Active Rules in Database Systems*. Springer. book.

Paton, N. W. and Diaz, O., 1999. Active database systems. *ACM Computing Surveys*, **31**(1):63–103. article.

Pattison-Gordon, E., Cimino, J. J., Hripcsak, G., Tu, S. W., Gennari, J. H., Jain, N. L., and Greenes, R. A., 1996. Requirements of a sharable guideline representation for computer applications. Technical Report SMI-96 -0628, Stanford University. techreport.

Rinderle, S. and Reichert, M., 2007. A formal framework for adaptive access control models. *Journal on Data Semantics IX*, **4601/2007**:82–112. LNCS 4601, Springer.

Rosa, M. L., Gottschalk, F., Dumas, M., and van der Aalst, W. M. P., 2007. Linking domain models and process models for reference model configuration. In *Business Process Management Workshops*, pages 417–430.

Rosemann, M. and van der Aalst, W., 2007. A configurable reference modelling language. *Information Systems*, **32**(1):1–23.

Schrefl, M. and Bernauer, M., 2001. Active XML schemas. In *International Workshop on Conceptual Modeling Approaches for e-Business, eCOMO*, Yokohama, Japan. inproceedings.

Shahar, Y., 2002. Automated support to clinical guidelines and care plans: the intention-oriented view. Technical report, Ben Gurion University, Beer Sheva, Israel. Commissioned by OpenClinical, 2002.

Shao, F., Antal, N., and Jayavel, S., 2004. Triggers over XML views of relational data. Technical report, Cornell University Technical Report. techreport.

Sistla, A. P. and Wolfson, O., 1995. Temporal conditions and integrity constraints in active database systems. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 269–280. ACM Press. inproceedings.

Snodgrass, R. T., Ahn, I., Ariav, G., Batory, D., Clifford, J., Dyreson, C. E., Elmasri, R., Grandi, F., Jensen, C. S., Käfer, W., *et al.*, 1994. Tsql2 language specification. *SIGMOD Rec.*, **23**(1):65–86.

Sql/Xml, 2003. the first edition of the SQL/XML standard. Technical report, published by the ISO as part 14 of the SQL standard: ISO/IEC 9075–14. techreport emansour.

Tansel, A. U., Clifford, J., and Gadia, S., 1993. *Temporal Databases: Theory, Design, and Implementation*. Benjamin-Cummings Pub Co, 1st edition edition.

Tatarinov, I., Ives, Z. G., Halevy, A. Y., and Weld, D. S., 2001. Updating XML. In *SIGMOD Conference*, pages 413–424.

Terenziani, P., Fabrizio, M., Gianpaolo, M., and Mauro, T., 2000. Executing clinical guidelines: temporal issues. In *AMIA*, pages 848–852. inproceedings.

EAMA, 2002. Best execution: Executing transactions in securities markets on behalf of investors. European Asset Management Association (EAMA), 65 Kingsway, London WC2B 6TD, United Kingdom.

van der Aalst, W. M. P. and ter Hofstede, A. H. M., 2005. YAWL: yet another workflow language. *Information Systems*, **4**:245–275.

van der Aalst, W. M. P., ter Hofstede, A. H. M., and Weske, M., 2003. Business process management: A survey. In *Business Process Management*, volume 2678/2003. Springer Berlin / Heidelberg.

van der Vlist, E., 2002. *XML Schema*. O'Reilly, 1 edition.

van Dongen, B. F., Jansen-Vullers, M. H., Verbeek, H. M. W., and van der Aalst, W. M. P., 2007. Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants. *Computers in Industry*, **58**(6):578–601.

Verbeek, H. and van der Aalst, W., 2006. On the verification of EPCs using t-invariants. Technical report, BPM Center Report BPM-06-05, BPMcenter.org,.

Votruba, P., Miksch, S., and Kosara, R., 2004. Facilitating knowledge maintenance of clinical guidelines and protocols. *World Congress of Medical Informatics (MedInfo)*, .

Wainer, J., Billa, C. Z., and Dantas, M. P., 2008. ST-guide: a framework for the implementation of automatic clinical guidelines. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1325–1332, New York, NY, USA. ACM.

Walmsley, P., 2007. *XQuery*. O'Reilly, first edition edition.

Wang, Y., Li, M., Cao, J., Li, Y., Chen, L., Lin, X., and Tang, F., 2006. An ECA-rule-based workflow approach for advance resource reservation in shanghaigrid. In *APSCC '06: Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing*, pages 421–426, Washington, DC, USA. IEEE Computer Society.

Weber, B., Wild, W., Lauer, M., and Reichert, M., 2006. Improving exception handling by discovering change dependencies in adaptive process management systems. In *Business Process Management Workshops*, pages 93–104.

Widom, J. and Ceri, S., 1996. *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann. book.

Wu, B., 1998. A test protocol description language: PLAN. Technical report, KCamp, Dublin Institute of Technology (DIT).

Wu, B. and Dube, K., 2001. PLAN: A framework and specification language with an event-condition-action (ECA) mechanism for clinical test request protocols. In *Hawaii International Conference on System Sciences (HICSS-34)*, page 140, Los Alamitos, California. IEEE Computer Society. inproceedings.

Yoshikawa, M. and Amagasa, T., 2001. XRel: a path-based approach to storage and retrieval of XML documents using relational databases. *ACM Trans. Inter. Tech.*, **1**(1):110–141.

Zaniolo, C., Stefano, C., Christos, F., and T., S. R., 1997. *Advanced Database Systems*. Morgan Kaufmann. book.

Zeng, L., Flaxer, D., Chang, H., and Jun-Jang, J., 2002. PLMflowdynamic business process composition and execution by rule inference. *Technologies for E-Services*, **2444/2002**:51–95. Springer Berlin / Heidelberg.

Zhen Hua Liu, M. K. V. A., 2005. Native XQuery processing in oracle XMLDB. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 828–833, New York, NY, USA. ACM Press. inproceedings.

Zoumboulakis, M., George, R., and Alexandra, P., 2004. Active rules for sensor databases. In *VLDB*, Toronto, Canada. inproceedings.

Appendix

A

The XML Schema of the AIM Specification Component

The AIM specification component consists of:

- Protocol Library
  - Global Rules
  - Protocol
    - Protocol Rules
    - schedule
      - schedule Rules

In AIM specification component, the rule consists of:

- Rule
  - Terms
  - Event
  - Condition
  - Action

*** The XML Schema definition for the protocol library**

```
<xsd:element name="protocolLibrary">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="protocols">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="pxsd:protocol" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="globalRules" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="rxsd:rule" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

*** The XML Schema definition for the protocol**

```
<xsd:element name="protocol">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="categoryID" type="xsd:token"/>
      <xsd:element ref="hxsd:header"/>
      <xsd:element name="schedules">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="sxsd:schedule" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="protocolRules" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="rxsd:rule" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
  </xsd:complexType>
</xsd:element>
```

The XML Schema definition for the header

```
<xsd:element name="header">
  < xsd:complexType>
    < xsd:sequence>
      < xsd:element name="releaseInfo">
        < xsd:complexType>
          < xsd:sequence>
            < xsd:element name="version" type="xsd:integer"/>
            < xsd:element name="institution" type="xsd:string"/>
            < xsd:element name="author" type="personDT" minOccurs="0"/>
            < xsd:element name="specialist" type="personDT" minOccurs="0"/>
            < xsd:element name="lastModeficationDate" type="xsd:date"/>
            < xsd:element name="validation" type="validationDT"/>
          < /xsd:sequence>
        < /xsd:complexType>
      < /xsd:element>
      < xsd:element name="didacticInfo">
        < xsd:complexType>
          < xsd:sequence>
            < xsd:element name="purpose" type="xsd:string"/>
            < xsd:element name="explanation" type="xsd:string"/>
            < xsd:element name="keyWords" type="xsd:string"/>
            < xsd:element name="citation" type="xsd:string"/>
            < xsd:element name="links" type="xsd:string"/>
          < /xsd:sequence>
        < /xsd:complexType>
      < /xsd:element>
    < /xsd:sequence>
  < /xsd:complexType>
< /xsd:element>
```

*** The XML Schema definition for the person and validation datatype**

```
<xsd:complexType name="personDT">
  <xsd:sequence>
    <xsd:element name="name">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="firstname" type="xsd:string"/>
          <xsd:element name="surname" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="email" type="xsd:string"/>
    <xsd:element name="contactNumber" type="xsd:token"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="validationDT">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="production"/>
    <xsd:enumeration value="research"/>
    <xsd:enumeration value="test"/>
    <xsd:enumeration value="expired"/>
    </xsd:restriction>
</xsd:simpleType>
```

*** The XML Schema definition for the schedule**

```
< xsd:element name="schedule">
  < xsd:complexType>
    < xsd:sequence>
      < xsd:element name="name" type="xsd:string"/>
      < xsd:element ref="hxsd:header"/>
      < xsd:element name="scheduleRules" minOccurs="1">
        < xsd:complexType>
          < xsd:sequence>
            < xsd:element ref="rxsd:rule" maxOccurs="unbounded"/>
          < /xsd:sequence>
        < /xsd:complexType>
      < /xsd:element>
    < /xsd:sequence>
    < xsd:attribute name="id" type="xsd:ID"/>
  < /xsd:complexType>
< /xsd:element>
```

*** The XML Schema definition for the rule**

```
<xsd:element name="rule">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="properties">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="ruleScope" type="ruleScopeDT"/>
            <xsd:element name="ruleType" type="ruleTypeDT"/>
            <xsd:element name="priority" type="xsd:integer"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element ref="hxsd:header"/>
      <xsd:element name="body">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="tsxsd:terms"/>
            <xsd:element ref="exsd:event"/>
            <xsd:element minOccurs="0" ref="cxsd:condition"/>
            <xsd:element ref="axsd:action"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
  </xsd:complexType>
</xsd:element>
```

*** The XML Schema definition for the term**

```
<xsd:element name="terms">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="term" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="title" type="xsd:string" minOccurs="1"/>
            <xsd:element name="type" type="termTypeDT" minOccurs="1"/>
            <xsd:element name="mapping2DB" minOccurs="0" >
              <xsd:complexType>
                <xsd:choice>
                  <xsd:element name="mapEvent" type="mapEventDT"/>
                  <xsd:element name="mapElement" type="mapElementDT"/>
                </xsd:choice>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="id" type="xsd:ID"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

*** The XML Schema definition for the event**

```
<xsd:element name="event">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="on">
        <xsd:complexType>
          <xsd:choice>
            <xsd:element name="absoluteTime" type="xsd:dateTime"/>
            <xsd:element name="relativeTime" type="relativeTimeDT"/>
            <xsd:element name="episode" type="episodeDT"/>
          </xsd:choice >
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
  </xsd:complexType>
</xsd:element>
```

*** The XML Schema definition for the event types**

```
<xsd:complexType name="episodeDT">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="term" type="xsd:IDREF"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="relativeTimeDT">
  <xsd:choice>
    <xsd:element name="onceOff" type="baseRelativeTimeDT"/>
    <xsd:element name="every">
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:extension base="baseRelativeTimeDT">
            <xsd:sequence>
              <xsd:element name="for" minOccurs="0">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="granularity" type="granularityDT"/>
                    <xsd:element name="timeLength" type="xsd:integer"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:choice>
</xsd:complexType>
```

*** The XML Schema definition for the event base Relative Time DT**

```
<xsd:complexType name="baseRelativeTimeDT">
  <xsd:sequence>
    <xsd:element name="granularity" type="granularityDT"/>
    <xsd:element name="timeLength" type="xsd:integer"/>
    <xsd:element name="beforeORafter">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="BAValue"/>
          <xsd:element name="episode">
            <xsd:complexType>
              <xsd:simpleContent>
                <xsd:extension base="xsd:string">
                <xsd:attribute name="id" type="xsd:IDREF"/>
                </xsd:extension>
              </xsd:simpleContent>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="granularityDT">
    <xsd:restriction base="xsd:token">
      <xsd:enumeration value="second"/>
      <xsd:enumeration value="minute"/>
      <xsd:enumeration value="hour"/>
      <xsd:enumeration value="day"/>
      <xsd:enumeration value="week"/>
      <xsd:enumeration value="month"/>
      <xsd:enumeration value="year"/>
    </xsd:restriction>
</xsd:simpleType>
```

*** The XML Schema definition for the condition**

```
<xsd:element name="condition">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="description" type="xsd:string" minOccurs="0"/>
      <xsd:element name="logic" minOccurs="1">
        <xsd:complexType >
          <xsd:sequence>
            <xsd:element name="simplePredicate" type="simplePredicateDT" minOccurs="1"/>
            <xsd:element name="compositePredicate" type="compositePredicateDT" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
  </xsd:complexType>
</xsd:element>
```

*** The XML Schema definition of the simple and composite predicate datatypes**

```
<xsd:complexType name="simplePredicateDT">
  <xsd:sequence>
    <xsd:element name="operand1" type="operandDT"/>
    <xsd:element name="operator" type="logicalOperatorDT"/>
    <xsd:element name="Operand2" type="operandDT"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="compositePredicateDT">
  <xsd:sequence>
    <xsd:element name="junction" type="junctionDT" minOccurs="1"/>
    <xsd:element name="predicate" type="simplePredicateDT" minOccurs="1"/>
    <xsd:element name="morePredicate" type="compositePredicateDT" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

*** The XML Schema definition of the operand1 and operand2 datatypes**

```
<xsd:complexType name="operandDT">
  <xsd:choice>
    <xsd:element name="termID" type="xsd:IDREF"/>
    <xsd:element name="getValue" >
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="of" type="xsd:IDREF"/>
          <xsd:element name="number" type="xsd:integer"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="value" >
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="amount" type="xsd:string"/>
          <xsd:element name="datatype" type="valueDT"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:choice>
</xsd:complexType>
```

*** The XML Schema definition of the simple datatypes**

```
<xsd:simpleType name="logicalOperatorDT">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="eq"/>
    <xsd:enumeration value="neq"/>
    <xsd:enumeration value="lt"/>
    <xsd:enumeration value="lteq"/>
    <xsd:enumeration value="gt"/>
    <xsd:enumeration value="gteq"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="junctionDT">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="and"/>
    <xsd:enumeration value="or"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="valueDT">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="string"/>
    <xsd:enumeration value="integer"/>
    <xsd:enumeration value="float"/>
  </xsd:restriction>
</xsd:simpleType>
```

*** The XML Schema definition for the action**

```
<xsd:element name="action">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="description" type="xsd:string" minOccurs="0"/>
      <xsd:element name="do">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="AIMQLxsd:AIM-QLAction" minOccurs="0"/>
            <xsd:element name="proceduralAction" type="proceduralActionDT" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
  </xsd:complexType>
</xsd:element>
```

*** The XML Schema definition for the procedural action**

```
<xsd:complexType name="proceduralActionDT">
  <xsd:sequence>
    <xsd:element name="SendSMS" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="mobileNo" type="xsd:integer"/>
          <xsd:element name="content" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="sendEMAIL" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="from" type="xsd:string"/>
          <xsd:element name="to" type="xsd:string"/>
          <xsd:element name="subject" type="xsd:string"/>
          <xsd:element name="content" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="invokeMethod" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
          <xsd:element name="parameters" type="xsd:string" minOccurs="0" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

B

The Author's Publications Related to this Ph.D.

- Wu, B., Dube, K., and Mansour, E., 2008. The motion picture paradigm for managing information: a framework and approach to supporting the play and replay of information in computerised information systems. In 10th International Conference on Enterprise Information Systems (ICEIS), Barcelona, Spain. Springer.

- Mansour, E., Dube, K., and Wu, B., 2007. AIM: An XML-based temporal and ECA rule language for managing complex information. In International RuleML Symposium on Rule Interchange and Applications (RuleML 2007), Orlando, Florida.

- Wu, B., Mansour, E., and Dube, K., 2007. Complex information management using a framework supported by ECA rules in XML. In International RuleML Symposium on Rule Interchange and Applications (RuleML 2007), Orlando, Florida.

- Mansour, E., Dube, K., and Wu, B., 2007. Managing complex information in reactive applications using an active temporal XML database approach. In 9th International Conference on Enterprise Information Systems (ICEIS), Funchal, Madeira - Portugal. Springer.

- Mansour, E., Wu, B., Dube, K., and Li, J. X., 2006b. An event-driven approach to computerizing clinical guidelines using XML. In In the First International Workshop on Event-driven Architecture, Processing and Systems (EDA-PS06), In conjunction with ICWS 2006 / SCC 2006., Chicago, USA. IEEE Computer Society.

- Dube, K., Mansour, E., and Wu, B., 2005. Supporting collaboration and information sharing in computer-based clinical guideline management. In 18th IEEE Symposium on Computer-Based Medical Systems (CBMS 2005), Dublin, Ireland. IEEE Computer Society.

Index