University of Oslo
Department of Informatics

Valid Time in a Model Driven Framework

Geir Myrind

Cand Scient Thesis

August 2002

ABSTRACT

This thesis presents a framework for modeling and developing temporal information systems. The framework defines a model driven development process which automatically transforms temporal system models into executable systems.

Temporal applications have semantics not represented in conventional modeling languages, and the main contribution in this case is a temporal extension to an existing UML profile. The UML profile is extended with temporal concepts and constructs to provide a temporal conceptual modeling language. Temporal features are introduced by model elements defined with valid time semantics, that is, the information model captures earlier and possible future states as well as the current state of entities. The approach is based on timestamping entities with valid time intervals to represent when states of an entity were valid in the modeled reality.

Based on the semantically extended models designed using the temporal profile the framework allows automatic code generation of temporal information systems . Thus, we provide model driven tool support for developing temporal valid time applications.

Acknowledgements

This Master Thesis is submitted in partial fulfillment of the Cand. Scient Degree in Informatics at the Department of Informatics, University of Oslo (UIO). The thesis work was conducted at SINTEF Telecom and Informatics. I wish to thank my supervisor Bjørn Skjellaug for his guidance and patience with me. Tomas Wangen and Gunnar Lunde receive thanks for useful and interesting discussions. Special thanks to Trude who gave me the final motivation to finish the thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> History is an illusion caused by the passage of time and time is an illusion caused by the passage of history.
>
> — *Douglas Adams*

In this chapter we state the purpose of this thesis. A brief introduction to the problem domain is provided and a statement of the primary goals is presented.

1.1 The Domain of Interest

During the last decade the software industry has evolved significantly. Although the advances in technology have made information management easier, there is still room for improvement. Many systems handle information with more complex semantics than simple, single state business objects. One example is systems with management of geographically referenced data, that is, typically systems with GIS components [83]. Geographically referenced in this context concerns data entities consisting of both temporal and spatial values. A variety of applications manages objects like estates, roads, buildings etc., which have properties related to time and space together with conventional attributes. All these properties combine to form a complete view and specification of a real world object. Conventional development techniques and modeling languages lack concepts to handle information of this type. As a mean to overcome the above deficiency, this thesis introduces a model driven approach to the design and development of information systems capturing the temporal aspects of such objects.

1.1.1 Temporal Data Management

Realizing that time is an inherent feature of all real world objects, it is not difficult to imagine the occurrence of time in applications. Temporal data

management is an area which has received wide attention by researchers the last two decades. Both research and practice have proved that capturing the time varying nature of data is a complex matter. No commercial temporal database, i.e. a database that supports some built-in aspect of time semantics[1], is available. Database application developers are forced to manage temporal information in an ad-hoc manner, building the temporal information logic into the application.

Most research has been concerned with the formal logic or the DBMS level of temporal databases. Focus has been on understanding the semantics of time varying information, a goal which is now fairly well understood [64]. Unfortunately, most users and developers are unaware of the results. Similarly, the outcome of research have had little or no impact on the availability of temporal database support. Developers could largely benefit from a temporal modeling and development framework, a topic which has received less concern [75].

1.1.2 Conceptual Modeling

Conceptual modeling is the art of representing the real world, often referred to as the universe of discourse or mini-world. The UML modeling language is one such technique defined by the OMG (Object Management Group) [50]. As an industry standard and a language with a high degree of flexibility, UML is a good choice for conceptual modeling. Traditionally, conceptual models were mainly used to represent the artifacts of computer systems, that is, to document the system at hand as inspiration for the application developers. The UML and its associated technologies have changed this, a new paradigm has emerged where the bridge between programming languages and conceptual models fades away. Model driven development is a methodology where all vital parts and aspects of the system under consideration are described with semi-formal models [46]. A formal model in this context is a model with specified semantics, structure, behavior and designed with a language having a well-formed definition, e.g. some UML profile. The shift to a model driven approach for application development has gained wide acceptance and is now also the focus of OMG in the form of the Model Driven Architecture (MDA) [67]. The COMDEF, which is a flexible framework for component development developed at SINTEF [66], supports a model driven development process using UML.

By taking advantage of this trend and exploiting the conceptual model of COMDEF, accurate models can be designed capturing the temporal aspects of the real world. If sufficient temporal semantics are present in a model, we can substantially help to simplify the development of temporal applications, e.g. by code generation as focused by this thesis. The UML, OMG, model

[1] Use of time related datatypes, such as e.g. Date, does not qualify as an aspect of time in this context

driven development and the COMDEF are presented more thoroughly in chapter 2.

1.2 Goal

Current modeling languages are too inaccurate in their expression of semantics to lead to an automatic generation of code in a software development process. A conceptual model should capture all vital aspects of the real world of interest, that is, properties as temporal aspects of objects should be represented in an expressive and meaningful way. Based on the lack of temporal concepts in existing modeling languages, we aim at extending the information modeling capabilities of the COMDEF framework with temporal semantics and notation for modeling persistent temporal objects. We have chosen valid time as the temporal aspect because the concept of valid time is the most referenced aspect in the modeled reality and manages past, present and future times. Valid time captures the real evolution of data managed by an information system, being more specific, valid time is defined as the time some piece of information, i.e. a fact, was true in the modeled reality [30].

Introducing valid time to COMDEF requires the design of a temporal data model. A temporal data model is constituted of the following parts:

- An ontology of time.
- Temporal concepts.
- A data model associated with the above.

The time and temporal model have significant impact on the expressiveness of the data model, similarly, the introduction of temporal concepts to a data model may take various forms. The COMDEF data model is defined as a UML Profile, which is a collection of extended and well-defined UML model elements for a specific domain. By extending the COMDEF UML Profile with temporal concepts and notation we propose a temporal UML profile. A variety of temporal data models have been proposed in literature, but few have examined the definition of a temporal data model to be used in a model driven setting. We have thus designed a valid time temporal data model represented by a Temporal UML Profile along the following guidelines.

- Extension conforming to the UML standard.
- Profile capturing well-accepted temporal aspects of valid time.
- Precise definition of temporal concepts.
- Expressive and rational temporal support.

- Simple and user friendly.

Aiming at these goals we establish a temporal UML profile in alignment with the requirements of the Model Driven Architecture. We advocate a general temporal data model where the main aim is to support the modeling of temporal information systems using the relational model. Utilizing the COMDEF we can design models having explicit temporal semantics and help the development of temporal applications.

1.3 Thesis Context

The thesis is written in context of different projects at Sintef. Main projects the thesis is related to are described below.

Geoman

Geoman was a project defined within the Dynamap project. The identification of a growing need for new technology capable of handling the large amounts of existing and future geographical data was the basis for Dynamap. As a subproject, the Geoman project had the task to find and introduce new ways of accessing and managing spatial, temporal and geographical features of information. In this context two main tasks was identified, new data structures and representational models has to be designed to more effectively represent spatio-temporal data and new mechanisms has to be defined to extract data from the new structures.

OBOE and COMDEF

Open Business Object Environment (OBOE) was an EU project as a cooperation between SINTEF and others where the main aim was to develop a framework and methodology for Business Object Modeling and deployment [47]. Sintef's role in the project resulted in the development of the COMDEF framework. Lately a new project called COMBINE introduces new concepts and develops a new framework bringing the base researched in OBOE further. Aim of COMBINE is to support model driven development in enterprises using components [15].

ST-tools

ST-tools was an internal project at SINTEF closely related to Geoman and the COMDEF framework. Using COMDEF as a conceptual modeling framework with strong code-generation facilities was the basis when the ST-tools project focused on extending modeling languages with spatio-temporal features. The aim was to model spatio-temporal data types at the conceptual

level and keep the data types down through the parsing layers so the implemented application will have conforming types with the conceptual model. To accomplish the aim, additional requirements in context of languages and models were needed.

1.4 Thesis Structure

The following describes the structure of this thesis.

Chapter 2 Background Introduces background information on concepts and technologies used in the thesis.

Chapter 3 Case A case model of a land information system (LIS) is presented. The aim is to identify shortcomings of conventional development techniques and modeling languages. Results of the chapter are the recognition of a requirement specification.

Chapter 4 Requirements Temporal requirements identified in chapter 3 are presented closer. Additional requirements for temporal data models and modeling are added and described.

Chapter 5 The Time Model and the Temporal Model The time and temporal model which the temporal data model is based on are presented.

Chapter 6 The Temporal Data Model The extensions to the COMDEF framework are described. The Temporal UML Profile which defines the temporal data model is presented.

Chapter 7 Case revisited The LIS from chapter 3 is presented again, this time using the new concepts introduced in chapter 6. A case application based on the revised case model is described .

Chapter 8 Evaluation and Discussion This chapter evaluates the temporal data model against the requirements defined in chapter 4. Discussion of results are presented.

Chapter 9 Conclusion and Further Work The fulfillments are evaluated and discussion is concluded, contributions are presented and further work proposed.

Chapter 2

Background

The main aim of this chapter is to give the reader insight into the domain in question. Technology, models, languages, tools and methods available are described along with the work done by others. The chapter is divided into two main parts. First we will introduce what we have defined as a modeling paradigm which is a general description of the recently emerged software development methodology. The second part introduces temporal data management and temporal modeling in special. At the end we present a few issues concerning Database Management Systems (DBMSs).

2.1 A Modeling Paradigm

During the last decade the process of developing software systems has changed significantly. New technologies have emerged and one common trend is the shift of focus from system implementation to system modeling. As new technologies and standards have been defined, much to the gratitude of organizations like the OMG (Object Management Group), are we looking forward to a new and better future world of software development. OMG is an international organization supported by hundreds of vendors, software developers and users. The aim is to establish guidelines and specifications to provide a common framework for application development [49]. In the next sections we introduce the modeling standards and technologies defined by the OMG and related work conducted by SINTEF.

2.1.1 UML

In the early nineties, a number of object-oriented modeling languages started to appear. The technique of modeling was invented as a result of the seeking for new and alternative ways of software analysis and design. When software systems became more complex, modeling languages gave developers the opportunity to model their problems using a visual view of the problem

domain. Models proved to be a helpful tool to simplify reality and understand the system being developed [8].

The different modeling languages had their features and shortcomings. An effort to construct one single language that converged the best practices in object modeling so far resulted in the UML (Unified Modeling Language). OMG officially adopted UML 1.1 as its object modeling standard in November 1997. Soon after users recognized the benefits of a common modeling language and today the UML is the dominant industry standard when it comes to modeling software applications [50]. As the de facto standard, UML has gone through several refinement processes. OMG performs continuous evaluation, each version is a collaboration between software groups. Partners are focusing on improving the language at each stage, at the moment of writing, the UML is on its way to version 2.0 [39]. The following quote is the definition of UML found in [50].

> The Unified Modeling Language (UML) is a language for visualizing, specifying, constructing and documenting the artifacts of software systems.

More rigorously, UML is defined by an abstract syntax, i.e., diagrams with associated well-formedness rules using a formal language and a description of the semantics in natural language [12]. All these components comprise the UML metamodel, which defines the language used to realize the visions of the quote above.

The UML has proved itself as a powerful modeling language applicable to many domains, but one single language can hardly address every domain perfectly. This introduced the notion of profiles. A UML profile is a package of UML related extended elements that capture domain specific features [50]. A set of extension mechanisms exists in UML to facilitate the construction of domain specific model elements. As the shortcomings of UML have been important for our work, especially the extension mechanism, the topic is described closer in section 2.1.3.

2.1.2 Model Driven Development

The last years have shown an even greater degree of complexity in software systems, the different requirements systems in enterprises are to fulfill are rising. Current modeling techniques are to general in their expressive power to fully support the complexity at hand. The result is more focus on system implementation instead of concentrating on system model design. Without a complete view of the system model being developed, the risk for mismatch between intended system model and implemented system is high.

Model driven application development is an answer to solve the above problem. A shift of focus to modeling of complete systems so that the whole

system structure is reflected at model level is necessary. The view is somewhat holistic; all concepts related should be expressed in the model. In this way software systems may automatically be generated from the model designed, and an implemented system corresponding to the intention is achieved [44].

This approach has several advantages. The result is not only more correct systems, but it also simplifies and reduces development time. By abstracting models away from platforms and languages, mappings to a variety of technologies are feasible. This is possible if all vital concepts of the related domain are represented in the model. Results are a stable information model that can be mapped to a variety of platforms [61].

COMDEF

COMDEF is a generic framework which facilitates modeling and development of component-based distributed systems. A definition of a reference architecture and tools used by the framework was developed to support a model driven form of development. Main parts of the framework are:

A reference architecture Concepts defined to support modeling of distributed systems.

A metamodel/UML profile A UML profile consisting of the concepts defined in the reference architecture.

A lexical language CML is a lexical language representing a one-to-one representation of the models modeled by the above UML profile.

Mapping facility Code generation tools based on the CML representation of models.

COMDEF was designed for developing distributed applications. Facilities for modeling distribution was introduced by the concepts *Service*, *UserService*, *Event* and *Entity*. These concepts are the main modeling elements defined in the framework. A set of dependencies was also defined to describe the different relationships between the concepts. In short, when developing systems with COMDEF, *Service* and *UserService* represents the server side and the client side distributed objects respectively. Event handling is defined by the *Event* concept. The *Entity* is used to model the real world objects in the system, i.e. the server side encapsulation of the information model. Typical usage is a client accessing a service component through a userservice which returns a reference to an entity object. As *Entity* objects are independent of *Service* components, the *Entity* allows us to model entities suitable for conceptual modeling in general. An important part of COMDEF is the code generation tools based on the CML representation of models. Mappings developed from COMDEF includes EJB and Corba,

another project has extended COMDEF for automatic GUI (Graphical User Interface) development [41].

Model Driven Architecture

Sintef has researched model driven application development during the last few years, e.g. in the OBOE project which resulted in the COMDEF [47], but it is just lately that this view has adopted global acceptance. Earlier the focus of OMG was CORBA, but as a result of the constant change in middleware and other technologies the OMG have decided to take a step up to solve the problem of technology integration [67]. The result is the MDA (Model Driven Architecture) which brings the ideology behind the COMDEF framework even further.

The MDA ties the different technology standards together, gives guidelines on how to use them and thereby stating how to construct, maintain and develop standardized models. MDA specifies an architecture where system functionality is separated from platform technology and implementation [44]. In this way, the architecture will be language, platform, vendor and middleware neutral. Platform independent models can the be mapped to a variety of platform specific models. OMG has already defined the standards that makes the MDA possible, that is, the UML which we have described and the MOF and XMI defined later in this chapter [67].

Metamodeling and the MOF

Modeling languages such as the UML provides modelers with defined concepts and notation to design models. The origin of the concepts and notation is defined in the metamodel, which defines usage of these in a model, i.e. metamodels are models of models, they define the information that can be expressed in other models [24].

Defining metamodels has proved to be a useful technique for a variety of reasons. Offering a higher level of abstraction provides a basis for understanding and describing the problem domain, thereby managing complexity. Metamodeling is an efficient method to express characteristics of different model elements, means are to provide common understanding of the modeled domain. Metamodels help users perceive the complex relationships of the various modeling elements and have proved to be the right way to express the abstract syntax of e.g. UML [12].

Meta Object Facility (MOF) is the standard for metamodeling and metadata repositories defined by the OMG [49]. The MOF have been used to define the UML metamodel and is consequently fully integrated with the UML. A small but expressive subset of UML is used by the MOF to define metamodels for various domains. MOF formally introduced the four-layered metamodeling architecture shown in table 2.1 [49]. Metamodels defined us-

ing the MOF, such as the UML metamodel, sits at the metamodel (M2) layer.

Layer	Model	Instance	Example
M3 Metameta model	MOF	metametaclass	“Metaclass“
M2 Meta model	UML metamodel	metaclass	“class“
M1 Model	User model	class	“Owner“
M0 System	User objects	Object	“Sam“

Table 2.1: Four layer meta-data architecture

Standards for metamodels and modeling languages are important when it comes to handling the complexity of software development. If data is to be interchanged between tools, applications, middleware and repositories, defined standards are necessary. Integration of metamodels across domains is required for integrating tools and applications during the software life cycle [42]. When extending the UML, metamodeling is helpful and important. Precise definitions of the extended model elements are vital. This way of using metamodeling is a great tool when describing for example UML profiles, where profiles are metamodels for models in a specific domain.

XMI

XMI is a key tool in a model driven environment. The XMI (XML metadata interchange) was adopted to the list of OMG technologies in February 1999 as XMI 1.0 [51]. The first intention with XMI was to provide a textual format and exchange of UML models, but it is now a general method for interchanging MOF based data. XMI is based on XML (Extensible Markup Language) defined by W3C in 1998 [79]. Since XMI is compliant with the MOF, it supports data at all levels, and interchange of metamodels, models and data is possible. Because of this, platform independent models can be interchanged between tools. XMI integrates three key industry standards, MOF, UML and XML, the best of OMG and W3C modeling technologies [51].

2.1.3 UML Shortcomings

As UML has become the de facto modeling language, several shortcomings and side effects of the UML have become evident. Problems discovered concern different areas. In our work the problems have appeared when trying to extend UML with new constructs. Introducing new features to UML is not an easy task, as the expressiveness of the extension mechanism is limited [11]. Three basic extension mechanism are defined in UML:

- Stereotypes: Singular properties that can be used to classify any UML model element. A stereotype can be either graphical (icon) or textual. Stereotypes are used to define specialized model elements based on a UML metamodel element. The problem with stereotypes is that they are singular only, i.e. a model element can only be stereotyped once, complex semantics are therefore hard to express.

- Constraints: Expressed in some textual notation, preferably formal as e.g. OCL [82], are static expressions regarding the behavior of model elements. OCL can be used to define constraints on any model element at any metalevel, the constraint applies to instances of the model element. E.g., specialized behavior of stereotyped elements are expressed using OCL.

- Tagged Values: Properties that can be applied to any model element. The tagged value consists of a name-value pair of textual strings. Different from attributes, tagged values can be viewed as metadata values denoting arbitrary intrinsic properties applied to a model element. Tagged values are user or tool interpreted.

Stereotyped elements inherit structural and behavioral features from its base UML metamodel element where constraints and tagged values are used to express difference from the base element. Use of stereotypes is a lightweight extension mechanism, the only problem is the singular usage restriction. The singular property is seen by us and others as a problem that seriously restricts the applicability of stereotypes. Another issue is that the UML metamodel is somewhat vague regarding the definition of stereotypes. Precision in models is important, ambiguous models lead to personal interpretations of a language.Modelers are given too much freedom, a stereotype must have well defined semantics which is a difficult task since the UML is not formally defined itself [3]. Use of the MOF to define a new UML metaclass is another solution, but this classifies as a heavyweight extension not desirable.

Support for a better extension mechanism should be a topic for the next revision of UML. The concern is also related to the possible growth of profiles, a view of future UML is as a family of UML profiled languages, each which its own domain specific features, e.g. platform specific models. As the family of languages evolve, we will end up with too many profiles and no infrastructure to handle changes and mappings between languages [12].

2.2 Temporal Modeling

During the last two decades, temporal data management has benefited from substantial research and a wide range of papers on the domain have been

written [31]. However, most research has been on defining a specific temporal model and its incorporation to existing databases. That is, the main issues of research have been at the logical or the implementation level. The introduction of temporal aspects into conceptual models have received less concern [54]. This section will introduce the domain of temporal data management in general with an emphasis on temporal modeling.

2.2.1 Temporal Applications

Time is an inherent concept in everyday human life. Experiences, facts, objects or events are all related to some kind of time. Software systems representing parts of our world are no exception, as most applications found are temporal in nature. Traditional databases contain only information of the current state in the modeled reality. Modifications by applications result in changes in the current state and usually the past state is deleted and lost. Retaining past states as a history would for many applications be of great benefit. Obviously, the business logic of an application may state that only current data needs to be stored, but again keeping historical states may be a critical feature for other applications [6]. Some applications even require access to possible future states [76].

The amount of areas and applications where time-varying information is present is vast. In general, every domain incorporates some kind of time in the handling of information. Temporal data management can be incorporated into every application. Needs are found in areas such as banking, finance, insurance, personnel recording and medicine. Another example is a land information application which may want to store and retrieve information about changes in land, variation of ownership or possible planned changes. Actually, to identify applications that do not handle time varying data is rather difficult [65].

Using conventional databases to develop temporal applications rely on time related data types in the database. Data may be timestamped using date attributes to represent a duration of time when the data had some property. Unfortunately, all temporal semantics of the application must be tailored at hand by application developer. The database provides no temporal support whatsoever. Often the process of defining temporal functionality is not handled properly due to the difficulty of managing temporal data. No standardized way of implementing temporal database applications is present. Another issue is the reinvention of time representation and structures for their manipulation which must be developed each time. Temporal functionality is hard to implement, difficult to maintain and can be performance demanding in certain cases [32].

The response from the research community regarding the above problem have resulted in a variety of temporal data models where the model has built-in support for capturing time varying information. Such models are

the topic of the next sections, we start with a brief description of some basic time concepts.

2.2.2 Time Basics

Time is a topic for philosophical discussion not to be elaborated over herein. The characteristics described below are simple properties and basic terminology for representing time in computers.

Humans see time as a stretching line from the infinite past to the infinite future. For databases the context is often the same, but time is naturally bounded by fixed sized data structures [65]. Mainly two structural models of time has been proposed in temporal logic [53]. In the linear model time advances from the past to the future totally ordered, while in the branching model time advances ordered until the present time and then divides into several possible future time lines. The linear model is common for general temporal applications, but when the outcome of evolution may have different alternatives, e.g. in planning, a branching time model is necessary. Ancient views conceived time as a cyclic structure [60]. A circular model of time may benefit application modeling recurring events [45].

Using the linear model, there are three different interpretations of the timeline. The discrete model of time is isomorphic to the natural numbers, dense models are isomorphic to the rationals, continuous models are isomorphic to the reals and contain no gaps [53]. One may argue which model fits reality the best, a topic scientists and philosophers have discussed for ages [55].

Another topic for discussion is the notion of points in time. Single points in time on an underlying time axis are termed instants, which have no duration [30]. But how should instants be represented in a database? An important term in this context is a chronon, defined to be the smallest unit of time in a computer system [30].

2.2.3 Temporal Models

Our main concern related to time is that of facts. Unlike events which can be said to occur at an instant, a fact is something that is valid over a duration of time [62]. To represent the duration of time when a fact has some property the fact must be associated with a value from a time domain. We say the fact is timestamped, a timestamp is a time value associated with some object, e.g. an attribute value, tuple or object [30].

A temporal model describe the time domains used, the temporal primitives available to be associated with facts, and possible operations on those. Main differences between temporal models are listed below.

- Dimensions supported

- Timestamp types
- Past or future time values

Time is dimensional, different temporal aspects may be associated with a fact. Two fundamental dimensions are the valid time and the transaction time of a fact. The valid time of a fact represents points in time when a given fact was, is or will be true in the modeled reality [31]. Timestamping facts with time values drawn from what we call the valid time dimensions denote when the facts were valid in the modeled world. The semantics imposed by the above states that all facts have a valid time by definition. Transaction time on the other hand, represents the time when a fact actually was/is resident in the database [33]. If a fact is associated with values from both dimensions, we have a bitemporal association between facts and time. A bitemporal relation inherits the advantages of both dimensions. It has the power to record both time varying data and retain earlier database states [53].

User defined time is sometimes also referred to as a temporal dimension. The term refers to facts associated with time values which are interpreted by the user only, in a temporal database the time values have no semantics except for the user [30]. A typical example is date of birth registered for a person. User defined time is represented in a database using time related data types.

Three different types of temporal primitives are used as timestamps in literature. A timestamp may be composed of a single instant, i.e. a point in time [30]. Intervals represent the time between two instants and is a pair of a starting and ending instant [35]. As a third alternative, temporal elements are finite unions of intervals [22]. Comparison operators on the temporal primitives in a temporal model may be defined, an example is the interval predicates defined by Allen [1], useful for reasoning over interval timestamped facts.

A valid time line is depicted in figure 2.1 recording owner history for a house entity. From time *t1* until *t2* Geoff is the registered owner, a change

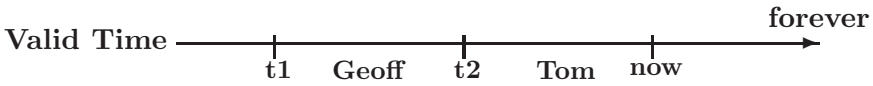


Figure 2.1: Owner facts and valid times

in ownership occurred at time t2 and Tom was registered as the new owner. The noun *now* is depicted as a value on the valid time line with the meaning that Tom owns the house until the current time. *Now* is simulating the current time as a moving target. Instead of updating currently valid timestamped facts at every instant or clock tick, *now* is used a constantly

varying variable equal to the current time value. The notion of *now* has received wide attention from philosophers for many years and its semantics are far more complicated than e.g. its counterpart in space *here* [14]. Such values classify as temporal variables and have been proposed in several temporal models [78]. Models which support future time values may use *Forever* as a valid time value, *forever*, equal to ∞, follows the last instant on the valid time domain [30]. Other variables are *beginning* and *UC* (until changed) [62]. Temporal variables are convenient, but also introduce some subtleties demanding careful consideration. The semantics of temporal variables and especially *now* has been extensively studied in [14].

Another topic is whether timestamped facts have values associated to exact points on the timeline or the values are relative to some other time or, e.g., *Now*. This introduces the distinction between absolute and relative time respectively [30].

2.2.4 Temporal Modeling Techniques

Above we introduced temporal concepts to be associated with facts in a database. In data models the representation of facts may take various forms. The lack of temporal support in databases has resulted in a variety of temporal data models. Early work was mainly based on using the relational model to support time. As the object-oriented paradigm emerged, temporal object models also started to appear [4].

We have described the non-adequate approach of using date attribute values to support time where all temporal semantics are left to the application developer. This led to the approach where a non temporal data model is extended to a temporal data model. Such models have built-in support to store, manipulate and query time varying information [76]. In the form of extensions to the relational model, data in schemas are timestamped and changes are made both to query language and relational algebra [62]. This extension approach is the main method to define temporal data models. Another option is to define a generic object model where all model concepts are given temporal semantics. An example of such a generalization approach of an object model is found in [69]. If a database supports ADTs (Abstract Data Types), a feature of object-oriented and object-relational databases, time can be implemented as an ADT having operations which can help building temporal semantics into applications. This approach is based on the extensible nature of object oriented systems, example models are found in [84, 70], and cannot be applied to relational databases.

Having described the approaches of defining temporal data models briefly, we turn to how time is associated with facts in the models.

Object/Tuple or Attribute Timestamping

In a valid time temporal model, facts are associated with temporal values denoted by timestamps. We now consider how facts are represented in a temporal data model and on what level the timestamps are applied.

Using the relational model, the main difference is the choice of attribute or tuple timestamping. The two different approaches have their pros and cons, models based on tuple timestamping retain the simplicity of the relational model and is the most common approach. Attribute timestamping on the other hand, is effective and space preserving since all information about an entity is stored in one single tuple [65]. In contrast, attribute timestamping does not obey the first normal form (1NF) and is therefore hard to implement in conventional databases [33].

We use the two tables in table 2.2 to illustrate the two approaches. The

House	Owner	Valid
31	John	1952 - 1975
31	Peter	1975 - 1991
31	Mary	1991 - now

(a) tuple timestamped

House	Owner
[1952-now] 31	[1952-1975] John
	[1975-1991] Peter
	[1991-now] Mary

(b) attribute timestamped

Table 2.2: Two Valid Time State Tables

two tables both contain the same information. A tuplestamped relation, as it appears in the model defined in [63] by Snodgrass is illustrated in the left table 2.2(a). Note, Snodgrass's model is actually bitemporal, supporting both valid and transaction time, for brevity the transaction time timestamps are not shown in table 2.2(a). Table 2.2(b) is an example of an attribute timestamped relation based on the model defined by Gadia and Nair in [23]. Table 2.2(b) introduces the concept of lifespan, a term so far not described, where the interval 1952-now represents the time house number 31 has existed in the modeled world. Lifespan represents the time an object is defined [30].

The main disadvantage of tuple timestamping is that an object is represented by one or more tuples in a database instance. Tuples belonging to an object must be identified by the same value at all times, if not, no mechanism is present to evaluate that two tuples are states of the same object. This introduces the notion of a time-invariant identifier.

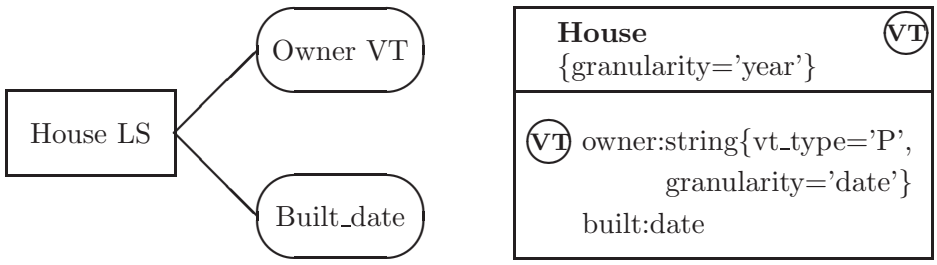
Above we assumed a temporal relational database as the storage of facts. The same concepts apply for object oriented systems, although the approaches used to support temporal aspects vary due to the inherent extensibility of such systems [69]. An object model allows a more natural incorporation of time and temporal aspects than the relational model [25]. For example, attributes are not restricted to atomic values which allows

a simple extension to support attribute timestamping. Examples where a time value is associated with whole objects are found in [34, 72, 20]. In these models, the object timestamping takes a similar form to tuple timestamping. Models where a timestamp is associated to each attribute usually have built-in support for capturing the lifespan of objects, examples of such models are found in [4, 54]. Three recent temporal data models are briefly presented in the next section.

Example Models

A review of a few temporal models are appropriate. Three models are presented, in forthcoming chapters these will be used as a reference and comparison to our work. Two of the models are based on UML, the third is a temporal extension based on the ER (Entity-Relationship) model. We are aware of a spatio-temporal visual modeling tool based on UML called Perceptory [58]. Main objective in Perceptory is the modeling of spatial databases, the temporal part is somewhat undocumented. Perceptory allows modeling of spatio-temporal objects using a variety of combinations of spatial and temporal symbols, using stereotypes, which can be applied at object and attribute level. Interestingly, the Perceptory tool allows generation of database schemas through a user interactive process. Due to the lack of temporal documentation and the specialized use of UML, Perceptory is not discussed any further.

A survey of temporal extensions to ER models exists in [27]. Based on deficiencies of current models found in the survey the authors proposed a new temporal ER model called TimeER [26]. TimeER is an extension to the EER model [19] and supports lifespan and transaction time of objects, valid and transaction time of attributes and relationships. An example figure of a TimeER diagram can be depicted in figure 2.2(a). In the diagram, LS is



(a) TimeER (b) TUML

Figure 2.2: A house modeled in TimeER and TUML.

used to denote that lifespan of the entity house is to be captured. Similarly,

valid time of the owner attribute is described by VT. Mapping algorithms of TimeER to the relational model are described in [28].

An approach using UML is found in the figure to the right 2.2(b), which is an example model using TUML [75]. TUML is a temporal extension to UML based on the TAU object model [36]. The TAU object model again is a compatible extension to the ODMG object model [9]. The Tau model supports the definition of valid, transaction and bitemporal object types. Attributes and relationships can be time timestamped according to the same dimensions. A history type is used to record the evolution of e.g. attributes over time. Each temporal object is associated with a lifespan. As depicted in the figure 2.2(b), the icon VT, which is a stereotype, is used to denote valid time lifespan of the House and valid time of the attribute owner.

TUML use stereotypes extensively to allow the incorporation of temporal support in UML. Similarly, the other known temporal extensions to the UML use a notion of stereotypes to establish their concepts. As described in section 2.1.3 the use of stereotypes are functional applied at class level, but its use on attributes are limited. The stereotype solution adopted by TUML is an approach we have ruled out.

The conceptual modeling language for spatio-temporal applications defined by Price et al. have also seen the limitations of stereotypes [56]. They have defined the Extended spatio-temporal UML by using new "stereotypes" that represent spatial and temporal properties by applying symbols. The symbols may be composite to denote different combinations of spatial, temporal and static values, and may be applied to object types, attributes and associations. Existence time of objects is introduced, a concept similar to lifespan but where attributes dependent on the existence time are grouped, while non-grouped attributes can have values independent of the existence time of the object. Such a grouping mechanism qualifies as an artificial UML construct. Detailed spatio-temporal semantics are defined in another special construct called the specification box. The specification box includes information on the time and temporal model to be associated with each temporal symbol.

Conceptual models must follow defined standards, in a model driven environment this is a key feature. In the case of extensions to the UML, all the models above use non-standard extension mechanisms not in alignment with the UML. Problems arise when using software tools that support the UML metamodel, tools will not be able to handle the non-standardized models. Users can benefit substantially of a conceptual modeling framework hiding complexity and at the same time provide the user with powerful concepts. In this way a user is able to correctly design a stable model corresponding to the real world, without being concerned with difficult modeling concepts and notation.

2.3 DBMS Technologies

Databases are the foundation for data management and provide applications with mechanisms for storage and retrieval of data. Main features of a DBMS are transaction handling, administration and security issues, constraints, recovery and concurrency control [19]. Traditionally, the relational database has been the commonly used architecture. Promises of relational databases are maturity and defined standards based on theoretical principles. Object-oriented databases on the other hand, lack tools, query languages and standards, but are better off for complex and user defined objects not supported in relational databases [37]. Examples of such objects are found in multimedia applications, Computer Aided Design (CAD) and GIS applications.

A variety of DBMSs are available. After the introduction of the relational model a range of databases where developed in the next decades. Commercial relational databases available today include products like Ingres, DB2, Informix, Sybase and Oracle [19]. Oracle is a large vendor product consisting of many features, for instance, a spatial module is developed to make spatial data management easy and natural for GIS applications [52].

The simple structure of the relational model and the lack of extensibility have led the major database vendors to extend their products. Informix is an example of a hybrid object-relational database where plugins from users or third party vendors can extend the functionality of the database. E.g. a temporal extension to Informix are described in [85] where a set of datatypes and routines enable temporal query support. A similar hybrid object-relational approach is the spatial module by Oracle mentioned above. Oracle Spatial supports two models to represent spatial extents; relational and object-relational.

As we have pointed out, no temporal database is available. A few prototypes have been implemented, among them a front end to the Oracle DBMS called Tiger. Tiger implements ATSQL which is a temporal query language based on SQL-92. An important feature of ATSQL is support for migration of time into existing applications [6].

2.4 Summary

We have described the advantages of model driven development compared to traditional software development techniques. The COMDEF has been presented as a framework supporting such a development methodology. Further, we have given an introduction to the domain of temporal modeling and its difficulties. In forthcoming chapter we present how we combine model driven development and temporal datamanagement to form a temporal conceptual model using COMDEF.

Chapter 3

CASE

The purpose of this chapter is to describe an application area within the domain of temporal data management. A case model is presented to identify shortcomings of current modeling languages and modeling techniques. The model should be able to describe these shortcomings, and be complex enough to show a necessary range of different applications. At the same time, the model should be easy to understand. Identified problems function as a base for defining requirements for a temporal extension to the COMDEF framework. The case model also serves as an example when introducing concepts throughout the thesis.

3.1 A Land Information System

A pure fictional but sufficient case model is presented, it is designed solely for the purpose of this chapter and does not represent existing systems or models. Land Information Systems (LIS) are important for many services such as land planning, and infrastructure development, resource management and a variety of other services [83]. The LIS case model is shown in figure 3.1. We briefly describe the entities depicted in the model.

A municipality represent a notably large piece of land that often represents authority and administration. A municipality has roads and parcels. Parcels again can have buildings of different types. For simplicity the category of buildings is divided into houses and commercial buildings. Both buildings and parcels can have owners, the owner may be a company or a single person. We devote our main discussion to the owner entity, since an important part of land information systems relates to changes in ownership. Ownership is subject to a variety of issues like contracts, inheritance, death, legal proceedings and accidents like fire or flood [83]. A requirement for a LIS is the capability to record variation in ownership over time.

The intended use of the system model is to provide a basis for a range of applications, which is completely genuine for a system model like this.

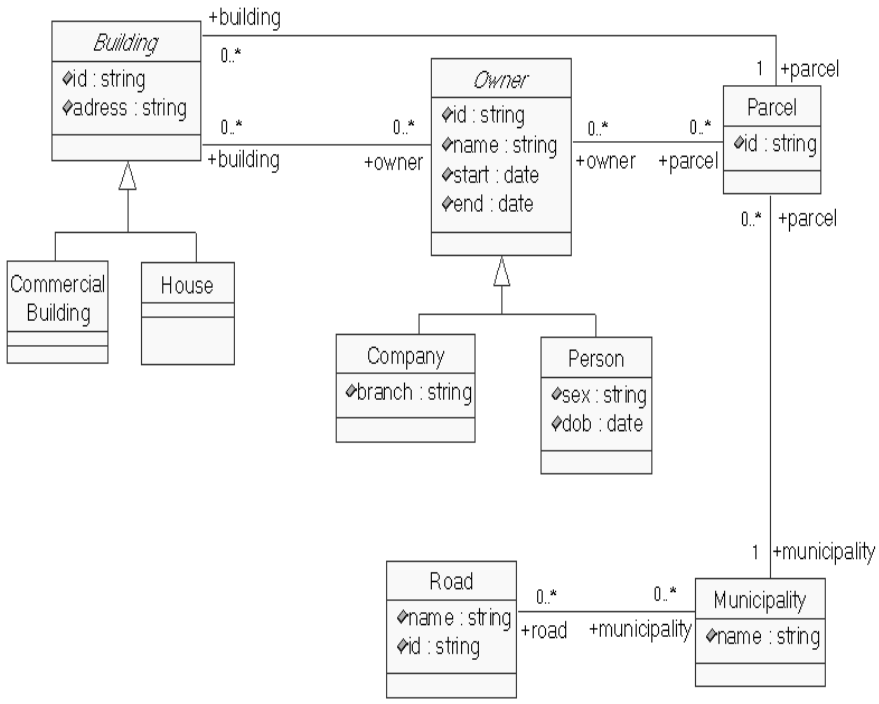


Figure 3.1: A snapshot casemodel

3.2 Application Areas Not Supported

The model is presented using conventional UML notation. All characteristics of the model elements are commonly available atomic data types. We have equipped the owner entity with a start and end attribute of type date. This is the simplest form of denoting time varying support for an entity, an example of a common ad-hoc solution, solely managed under user-control. The date attributes can be utilized to denote temporal semantics of some kind. However, built-in support for interpreting the information of the two date timestamps is not present. The model designer would probably know the intention, but different and ambigous user interpretations are easily described.

Imagine a database instance given by the case model, a variety of facts can be deduced, but they will all be facts based on snapshot semantics. No history of the changes in e.g. ownership are recorded in a practical way. For a user it would be convenient if one could query entities with respect to their temporal properties, i.e. view entity data at different points in time. Below are a few examples of temporal application queries we may want to execute,

but cannot easily be accomplished because of the lack of concepts in the model. An extract of a municipality and its variation over two decades is illustrated in figure 3.2 as a reference to the queries. We imagine that a

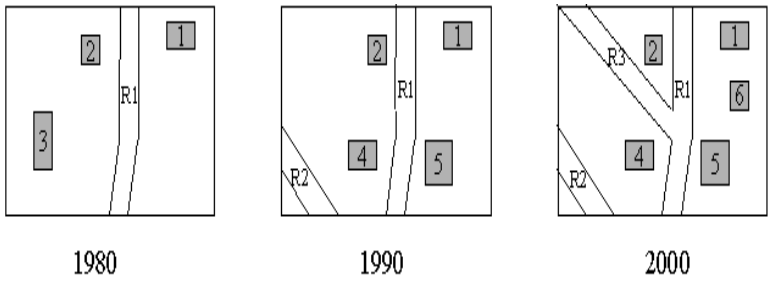


Figure 3.2: Variation of land and ownership

municipal council wants to keep record of the owners of a given building.

- Get all owners of building number 3 at all times?
- Who owned building number 3 before 1970?
- When was a commercial building first built on parcel 23?

A real estate company can have a variety of queries about the houses.

- When and how many times has a given building changed owner?
- When did John own house number 2?
- Which houses has Mary owned during her lifetime?

An insurance company may have the following question.

- Who owned house number 3 before Peter?
- Who owned house number 2 when Peter bought house number 3?

It is evident that the case model has to be extended with additional information to fulfill the requirements of such queries. Temporal aspects need to be expressed in a meaningful to make an implemented database instance capable of returning answers to queries as the ones above.

3.3 Limitations and Requirements Identified

The case model has no capability of representing properties related to the requirements of a LIS, i.e. objects varying in time and the representation of spatial extents. A complete model of a LIS should express all desired

properties and semantics in a visible and understandable way. At the same time the model should be complete in respect to the implementation of the system. Users should have the opportunity to concentrate on modeling the information model and not on designing sophisticated constructs hard to comprehend. What we need are new concepts to denote new constructs with defined semantics and behavior in the model. The following sections describe requirements found necessary for extending the case model to a more specific model aimed at developing a LIS.

3.3.1 Database Support

We need a notion to define that an entity in the model is by definition persistent and reflects the characteristics of common database entities. This means that a new concept which introduces database specific constructs must be defined. Below are the constructs we define as vital characteristics of entities which are to be mapped to relational database schemas.

Primary Key

In a relational database a primary key is used to uniquely identify tuples in a relation [19]. UML has no notion of key attributes. For generality we state that each database entity is required to have a primary key defined where the key's attribute is user defined.

Index

Efficiency of queries on large relations are greatly improved by attribute indexing [19]. Indexing time varying relations are important since they tend to grow large due to their historical nature [40].

3.3.2 Valid Time of Entities

An important feature of e.g. a LIS is the capability to record changes in land and ownership over time, because entities are subject to change. The history of an entity is therefore required to be stored in the system. What we need are new concepts to define temporal entities which have the necessary constructs to support changes over time. The opportunity to store both historical and future states of entities are important. It must be possible to denote a timestamp value recording the time when the properties of an entity in the model was true in the modeled reality. More specific requirements for valid time support for entities and how this is formed in a data model are presented in chapter 4.

3.3.3 Domain Specific Data Types

A new set of data types aimed at representing temporal and spatial values are necessary. Often we want to register some information related to time, like date of birth of an owner. Such temporal information is what we usually call user defined time, that is, the semantics have no further meaning than the one interpreted by the user [30]. Similarly, spatially referenced objects need data types representing types of spatial values such as points, curves and regions [59]. Research done in [81] has proved that data types are sufficient to represent spatial extents.

3.4 Summary

This chapter introduced a fictional case model. Its purpose was to outline shortcomings of current modeling techniques for spatio-temporal domains and thereof identify requirements for a modeling language. We recognized a need for database specific properties, valid time support for entities and new datatypes. Our main concern is the temporal aspect of the domain, that is, the requirement of built-in valid time support for entities. This topic is elaborated on in chapter 4 where we present requirements for temporal data models in a model driven setting.

Chapter 4

Requirements

The case model in the previous chapter identified shortcomings of conventional modeling languages and the necessity of new modeling concepts was stated. From a temporal perspective the main requirement was valid time support for entities. Based on the modeling paradigm and the temporal modeling techniques defined in chapter 2, we recognize the need to define a temporal data model supporting the concepts of model driven development. These two topics form, in this chapter, a requirement specification for a temporal modeling language. These requirements serve a guiding and evaluative purpose for our work.

The underlying foundations of temporal data models, that is the time model and the temporal model used, have a great impact on the expressive power and usability of a model. This is the topic of the first section of this chapter where we introduce basic temporal requirements regarding time and temporal models. The second section continues with temporal modeling requirements, which are divided into two categories; requirements for temporal data models in general and requirements for a modeling language in a model driven environment.

4.1 Basic Temporal Requirements

A temporal data model requires a precise and complete foundation. Below we present general criteria found in literature regarding time and temporal aspects required by temporal applications.

4.1.1 Models of Time

To be able to represent and reason over time varying information in a given context, a definition of time itself is necessary. The ontology of real world time itself has many facets, but for computer environments this diversity of time must have a precise definition. Requirements set by applications

vary, the choice of time model do have restrictions on the semantics to be represented in applications. Requirements for different models of time are listed in table 4.1.

R Time Model	Requirement
RT1	Linear
RT2	Branching
RT3	Discrete
RT4	Continuous
RT5	Infinite
RT6	Circular

Table 4.1: Time model requirements

Linear (RT1)
A linear model of time is the most common approach in literature, it is also a view of the time line in alignment with the general perception of real world time [26, 35].

Branching (RT2)
In contrast to linear time, a branching view of time may be of benefit to some applications. A branching time line appears when there is a need to reason over possible alternative futures, a topic for prediction applications such as forecasting [76, 25].

Discrete (RT3)
For most applications a discrete view of the timeline is adequate [65]. Real world time may be perceived as continuous, but for general database applications a discrete line is sufficient and implementation wise necessary [63].

Continuous (RT4)
Compared to a discrete view of time, some domains may benefit from a continuous model of time, e.g. real-time monitoring systems, hybrid systems and in maths or physics [10]. A continuous model of time is used to model moving spatial objects in a database in [21].

Infinite (RT5)
Some models explicitly define time boundaries, e.g. the TUML [75]. A bounded timeline is often a pragmatic choice since time values must be stored in data structures which are limited in size [65].

Circular (RT6)
A circular or periodic time is of benefit to some applications, e.g. in office automation or scheduling environments data may refer to periodic time, i.e. recurring events that happen every week, month or year [56, 45, 76].

4.1.2 Temporal Models

Based on the underlying time model a temporal model defines concepts identified to be useful when associating time with facts. The most common requirements found in literature regarding temporal models are listed below in table 4.2.

R Temporal Model	Requirement
RTM1	Valid Time
RTM2	Transaction Time
RTM3	User Defined Time
RTM4	Beginning
RTM5	Now
RTM6	Forever
RTM7	Until Changed
RTM8	Instant TimeStamp
RTM9	Interval TimeStamp
RTM10	Temporal Element TimeStamp
RTM11	Absolute
RTM12	Relative
RTM13	Interval operators

Table 4.2: Temporal model requirements

Valid time (RTM1)
Capturing the information history or the evolution of facts in a system is important for many applications. Valid time is defined for most of the temporal models found in literature, e.g. [65, 26, 75].

Transaction Time (RTM2)
Transaction time is supported in many models, usually a model supporting transaction time also supports valid time [75, 26, 76]. Transaction time is required by applications when it is crucial to retain past database states [32].

User Defined Time (RTM3)
In order to support user defined time a set of date times is required, i.e. time related data types must be available the user. Support for such types in a model is usually a simple requirement to fulfill since the implementation is

external and the semantics independent of the data model itself [27].

Beginning (RTM4)
Beginning is a special time value denoting the first point in the valid time domain [30]. The value is used by a variety of temporal models, e.g. the temporal extension to UML TUML defined in [75].

Now (RTM5)
Instead of updating the current time for a timestamp for every point in time, a task unpleasantly hard in a database, the current time is conveniently represented by a temporal variable. The special value *now* is used as a function always returning the current time and has been extensively used in temporal models [14, 65].

Forever (RTM6)
Forever and equivalent values as ∞ represents the value following the largest value of time for the valid time dimension [30]. *Forever* used by the temporal data model TUML to define the upper boundary of time [75].

Until Changed (RTM7)
Until Changed is a third type of special value. One option is to use *Until Changed* as a substitute for *now* where the intention is that a fact is valid until we know more. Other models use the variable as a special transaction time marker to denote end times of timestamps in transaction time models, e.g. in [26].

Instant TimeStamp (RTM8)
Instants or single points in time are one of three main temporal structures that can be associated with facts. Facts timestamped with an instant is usually assumed to be valid at the given point in time which represents an event based model [75]. A timestamp representation based on instants is used to represent the valid time of facts in e.g. [26].

Interval TimeStamp (RTM9)
Associated with facts, intervals represent a duration of time when a fact has some property. Most temporal models use intervals to denote the times a given fact was e.g. valid in the modeled world [57, 75, 76].

Temporal Element TimeStamp (RTM10)
Facts may be associated with temporal elements, i.e. finite unions of time intervals [22]. The TimeER model defined in [26] use this approach. Similarly, a temporal element is used to represent lifespan in many data models [20, 56].

Absolute (RTM11)
Absolute time refers to whether time values of facts are referenced to exact points on the time line, i.e. anchored points in time [53]. Conventional temporal models all support absolute time values, most applications requiring a historical dimension wants to associate facts to exact time points [38].

Relative (RTM12)
In models supporting relative time, time values associated with facts may not reference exact time points on an underlying timeline [53]. Relative models of time have been popular in the domain of artificial intelligence where the modeled world often contains relative temporal knowledge, an example is found in [38].

Interval Operators (RTM13)
Reasoning over the temporal primitives, the relationships between them may take various forms. Widely recognized are the temporal comparison operators defined by Allen [1]. Most temporal models provide interval comparison operators, either as functions operating on the datatypes or most preferably as built in operations in a query language [65].

4.2 Temporal Modeling Requirements

In the case chapter we identified a need for temporal concepts and constructs in the model. A temporal data model must be defined to allow the incorporation of such concepts. The requirements in this section are divided into two parts, first topic is temporal data models in general while the latter concerns requirements for such models in a model driven environment.

4.2.1 Temporal Data Models

Designing a temporal data model is not an easy task and defining a single model capturing all desired features has proved to be difficult, if not impossible [31]. This section presents different requirements for temporal data models described in literature. Timestamps below are assumed to be valid time timestamps, although the same requirements apply to transaction time data models.

Lifespan of Entities (RDM1)
The lifespan of an entity is the time the entity exists in the modeled reality. If lifespan for an entity is supported then a model has built-in support for capturing the times when entities exist. This feature is supported in e.g. TimeER, which is a temporal extension to the ER model [26].

R Data Model	Requirement
RDM1	Lifespan of entities
RDM2	Timestamped Entities
RDM3	Timestamped Attributes
RDM4	Timestamped Associations
RDM5	Time-Invariant Identifiers
RDM6	Granularity

Table 4.3: Temporal data model requirements

Timestamped Entities (RDM2)
Some models timestamp whole entities while other models consider the time varying nature of each attribute separately [4, 71]. The two approaches has it pros and cons, a topic we described in chapter 2. Example entity timestamped temporal data models are found in [72, 34].

Timestamped Attributes (RDM3)
Advantages of attribute timestamping are that changes in a single attribute only does not imply a change for the complete state of an object [75]. However, the approach is harder to implement than timestamping entities only, especially if the relational model is the underlying architecture. Example models supporting attribute timestamping is the TUML [75] and a spatiotemporal conceptual model called MADS [54].

Timestamped Associations (RDM4)
Semantics of associations between temporal entities may take various forms. Static associations may depend on the timestamps of the participating entities. Another solution is to timestamp the association itself, assigning the association time varying behavior independent of the entities. Models that explicitly capture temporal aspects of associations are found in [75, 54, 26, 56]. Note, associations in this case are structural relationships, specifying that e.g. an entity is connected to another.

Time-Invariant Identifiers (RDM5)
The concept of primary keys serve well as unique identifiers for snapshot databases. When entities vary in time, the notion of identifiers must be revisited. Two options are present, an identifier can uniquely identify an entity at all points in time, or an identifier can represent different entities at different points in time [54, 76]. E.g., a primary key value of an owner may be associated with different owners at different times or at all times identify a single owner. The most common assumption is the latter, a time invariant key is necessary to identify entities in a database [55]. Applications exist

where reuse of identifiers is convenient though, similarly, it is not always the case that an identifier value is constant over time [13].

Granularity (RDM6)
The granularity of timestamps associated with facts denotes the size of the time unit used, i.e. the chronon. The granularity of timestamps may differ from application to application, e.g. seconds, days, months or years. Some models allow the modeler to explicitly specify the timestamp granularity [75, 27].

4.2.2 Model Driven Development

Our aim is to design a temporal data model in a model driven framework within the frame of the Model Driven Architecture. Additional requirements to the temporal data model and its notation are therefore necessary. Table 4.4 below presents requirements for a temporal data model to be used in a model driven environment.

R Model Driven	Requirement
RMD1	Methodology Support
RMD2	UML Profile
RMD3	Interpretale Mapping from Graphical Notation
RMD4	Platform Independent
RMD5	Metamodel of Time and Temporal Concepts
RMD6	Simple and Expressive

Table 4.4: Model Driven Development requirements

Methodology Support (RMD1)
Methodology support refers to whether a standard modeling notation is used. Conforming to the standards is important for communication, interchange and integration of models. An example model not fulfilling the requirement is the MADS model [54] which uses a non-standardized notation based on a hybrid ER/OO model.

UML Profile (RMD2)
A UML Profile is a collection of enhancements of UML model elements for a specific domain. It is important that the extended elements have a consistent definition and are compliant with their reference metamodel, that is the UML or another profile. This is only achieved by a well defined metamodel with associated constraints and well formedness rules. Models based on metamodels not in alignment with the UML will not be applicable in an MDA environment [46]. A profile should also express guidelines on usage

and specify transformation rules of models.

Interpretable Mapping From Graphical Notation (RMD3)
The model should be interpretable from a graphical notation. Concepts defined in the model must be recognized by tools to maximize the code generation abilities. The usability of a temporal modeling language is greatly improved if a user can concentrate on modeling the problem domain with simple concepts and not have to be bothered with developing complex code and schema definitions. One of the goals of a mature MDA environment is generation of complete systems [67].

Platform Independent (RMD4)
A Platform independent model (PIM) is a language and technology neutral UML model. Some aspects of domain specific features can be present, i.e. profile concepts, examples are notion of persistence or ,as in our context, temporal aspects [61]. A PIM should, as the name specifies, be abstracted away from all platform-specific details.

Metamodel of Time and Temporal Concepts (RMD5)
In a model driven environment all semantics of a domain should be described in a model. This is only possible if all vital parts are expressed using a defined modeling language. A metamodel of time and temporal aspects should therefore be modeled to complete the UML profile. A metamodel for time is used to define temporal primitives in the temporal ER model [77].

Simple and Expressive (RMD6)
A requirement for temporal modeling languages is that models designed should be simple and expressive. Temporal aspects often clutters the notational design. Another issue is that some models wants to capture every concept possible and thereby only confuse users with a load of difficult notions. A temporal modeling language should be simple and user friendly and at the same time have enough expressive power [68].

4.3 Summary

We have presented requirements for time models, temporal models and for temporal data models in a model driven environment. Based on these requirements a time model and a temporal model are defined in chapter 5, and a temporal data model realized by a Temporal UML profile is defined in chapter 6. The fulfilled requirements are listed in their respective chapters, and an overall evaluation of our work is presented in chapter 8.

Chapter 5

The Time Model and the Temporal Model

In this chapter we define the general time concepts that form the basis of the temporal data model defined in chapter 6. We describe a model of time and define a temporal model. The definitions herein are based on the requirements for such models described in chapter 4 and fulfilled requirements are stated at the end. The aims of the temporal model are to define the basic concepts related to valid time and the temporal primitives to be associated with facts. A UML representation of the defined concepts is modeled as a foundation for the temporal data model to be defined in chapter 6.

5.1 The Time Model

This section focuses on time itself, the nature of the time domain and its properties as we present a generic model of time which can be used in an ordinary temporal data model. General time aspects are presented in chapter 2, but in order to ensure clarity some of the concepts are repeated. The time line may be seen as continuous or dense, but a computer system must necessarily have some sort of discrete encoding of time [18]. In agreement with the temporal database community, we define the time line to be discrete. We define a single point on the time line to be an instant. Instants have no duration, different from the concept of a chronon, defined to be the smallest non-decomposable unit of time in the model [30]. No specific boundaries on the time line are defined, although the beginning of time is assumed to start at the "Big Bang" and end with the possible "Big Crunch" following common cosmological arguments. Our definition of the time line $\mathcal{T}$ is therefore an infinite set of instants where each instant is separated by a single chronon. The following ordering properties for the time line apply, for two instants $t_i, t_j \in \mathcal{T}$, either $t_i \leq t_j$ or $t_i > t_j$. Based on the above defini-

tions, we state that $\mathcal{T}$ is equivalent to some subset of the integers. Intervals are used to represent durations of time, a time interval is defined to be the time between two instants [30]. To help understand the different concepts the relationship between instants, chronons and intervals are shown in figure 5.1.

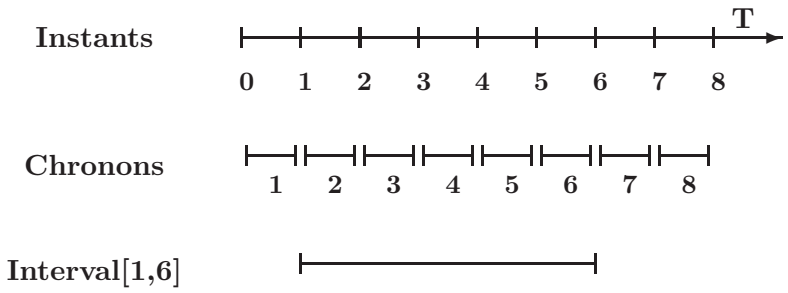


Figure 5.1: The relationship between instants, chronons and intervals

A closer definition of intervals are found in the next section when we introduce the temporal primitives to be associated with facts.

5.2 The Temporal Model

The former section described a general model of time. On this basis we define the temporal dimensions and type of timestamps to be used in the temporal data model. For temporal models there are two main aspects of time, the valid time and the transaction time of facts. Our main concern has been valid time and is therefore the only dimension supported in this model. We continue with a definition of valid time intervals and describe a set of operators.

5.2.1 The Valid Time Dimension

To store time varying information we introduce the concept of valid time to represent when a given fact is, was or will be valid in the modeled world. The valid time of a fact is denoted by associated time values from the valid time dimension. We define the time domain for valid time as follows:

$\mathcal{VT} = \{t_0, \ldots, t_i \ldots, now, \ldots\} \cup \{forever\}$, $t_0 \leq t_i < forever$. Characteristics to that of $\mathcal{T}$ apply, that is, according to the ordering and discreteness of the valid time domain $\mathcal{VT}$:

$t_k = t_0 + k$ and $t_k = t_{k-1} + 1, k \geq 1$ and $t \in \mathcal{VT}$

Two special values appeared, the value *now* is a special temporal variable always denoting the current time [14], and *forever* is defined as the instant following the last valid time instant on the valid time domain [30]. Semantics is that some valid times, i.e. valid time instants, are expected to be in the past and some may be in the future.

5.2.2 Valid Time Intervals

Facts in a temporal data model are timestamped with values from a temporal domain. A timestamp can be represented with a timepoint, interval or a temporal element. Points in time or instants cannot represent durations and it is e.g. difficult to decide the relationships between instant timestamped facts [1]. Temporal elements are on the other hand hard to represent in underlying architectures restricted to atomic data values such as the relational model. Intervals were thus chosen as the timestamp type for facts.

Above we introduced intervals as the time between two instants. Intervals are an encoding of a set of contiguous instants in time and is represented by the starting and ending instant. A valid time interval is an interval defined over the valid time domain, representing the time between two valid time instants. We define a valid time interval to be a $VTInterval = [t_i, t_j]$, in a closed-closed fashion, where $t_i \leq t_j$, $0 \leq i \leq j$ and $t_i, t_j \in \mathcal{VT}$. A $VTInterval$ of type $[t_i, t_i]$ defines the single instant $t_i \in VT$. The representation of intervals may have some pragmatic differences regarding whether a closed-closed or e.g. closed-open are used, but since the time line is discrete this is of no significance. Further we define the set of all valid time intervals by $I(\mathcal{VT}) = \{[t_i, t_j] \mid t_i, t_j \in \mathcal{VT} \wedge t_i \leq t_j \wedge 0 \leq i \leq j\}$.

A fact may be a single value, a tuple or an object. E.g. in a relational database, facts are recorded by tuples and each tuple may be associated with a timestamp. In our model facts are timestamped with valid time intervals. A duration of time represented by a valid time interval denotes when the fact was valid in the modeled reality.

5.2.3 Interval Operators

To fully exploit the promise of temporal reasoning, operators that can efficiently compare intervals of time in different ways are necessary [1]. Temporal analysis is important, many applications aim to analyze time varying data for different purposes. E.g. in a LIS, changes in urban structure over a period of time can be recorded and analyzed. The knowledge extracted may be used for different planning or prediction purposes [29]. We define a set of comparison operators on the valid time intervals. A pair of intervals can have one of 13 relations between them. The temporal comparison operators defined by Allen captures these relationships [1]. The relationships between

two intervals, a and b, are shown in figure 5.2.3. Only the seven basic operat-

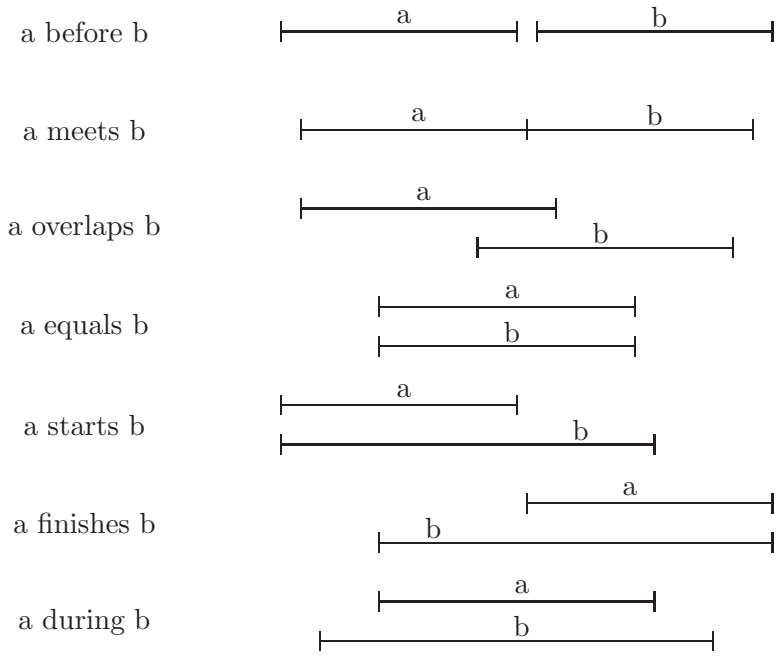


Figure 5.2: The possible relationships between two intervals

ors are shown, the remaining six are the inverse of the others. The operation equals has naturally no inverse operation. The description of each operator should be easily understood by the names and the figure. In chapter 6 the operators are defined explicitly when we introduce the temporal data model.

5.2.4 User Defined Time

User defined time is an uninterpreted temporal domain, usually supported by date and time attributes, where the semantics are known to the user only. In a temporal model user defined time is supported by a set of time related data types. The available data types in COMDEF are sparse, an extension to COMDEF to support new data types for user defined time are presented in chapter 6.

5.3 Time and Temporal Concepts Realized in UML

The former sections described a model of time and a temporal model. This section describes the realization of these concepts using UML. UML itself has no notion of such concepts, neither has the COMDEF. To introduce

temporal concepts to COMDEF we define a metamodel of time and temporal concepts using UML to be used as a part of the forthcoming Temporal Profile. The UML metamodel is simple and general, but defines concepts helpful for designing the COMDEF temporal data model. A conceptual model of the temporal model defined above is shown in figure 5.3. All concepts are described in the model, where the most important aspects are the valid time dimension and the valid time interval. The elements in the

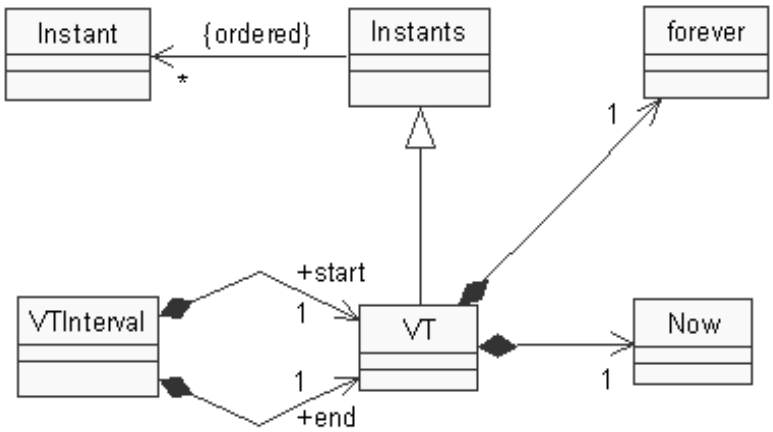


Figure 5.3: Temporal model Realized in UML

model are used as concepts for deriving the metaclasses for the Temporal Profile. Note that these concepts are abstract, they are not specifications of actual modeling elements in the forthcoming Temporal profile. Each element in the model is described in more detail below.

Instant

Instants, or single points in time, are represented by the Instant model element and can be depicted in the model. Instant is an equivalent concept to the instant defined in the time model.

Instants

Instants denote a set of Instant elements and is equivalent to the time line $\mathcal{T}$. The Instants model element is defined by an infinite ordered set of Instant elements, denoted in the model by a relationship to Instant having infinite cardinality. The constraint *ordered* on the association states that Instants are an explicitly ordered set of Instant elements. Note, from the definition of $\mathcal{T}$ defined in section 5.1, we state that the set of instants, i.e. Instants in this case, is isomorphic to some subset of the integers.

VT

This is the notion of the valid time dimension $\mathcal{VT}$, in the figure VT is a subtype of Instants. That is, similar to Instants, VT represents an infinite set of ordered instants, termed valid time instants for VT. Differing from Instants, VT have the two special temporal values of Now and Forever as associated elements. Note the singular properties of the associations, Instants have only one Now and one Forever. Now always denotes the current instant, while Forever represents the last instant on the valid time timeline.

- now = t in the sense that t = systemclock current time, $t \in VT$.
- forever = t' such that $\forall t \in VT \mid t' > t$.

Using OCL we constrain VT to be an ordered set of valid time instants.

```
Context VT INV OrderedDiscrete
          Sequence {0..{now..forever}}
```

VTInterval

VTInterval represents a valid time interval as defined above and represents the time between two valid time instants. VTInterval is realized by two associations to VT, denoted *start* and *end* as depicted in the model. Namely, a VTInterval consists of a start and an end valid time instant from VT. A constraint expressed using OCL for VTInterval follow. The *start* must precede the *end*, at the same time *start* cannot be equal to Now and must be less than Forever.

```
Context VTInterval INV intervalconstraint
               self.start <=self.end and
               self.start <> Now and
               self.start < Forever
```

5.4 Summary

In this chapter we have defined a general model of time and a temporal model. The main topic has been the definition of the valid time dimension and its related properties. A metamodel of time and temporal concepts have been modeled, in chapter 6 the model is combined with COMDEF to form the temporal data model. Requirements from chapter 4 fulfilled in this chapter are described briefly below.

RT1 Linear
A total ordering of the time line is defined, time advances from the past to the future in a linear fashion.

RT3 Discrete
The timeline is isomorphic to some subset of the integers, thus discrete.

RT5 Infinite
An infinite time line has been defined.

RTM1 Valid Time
The temporal model supports valid time.

RTM4 Now
Now is introduced as a current time temporal variable.

RTM5 Forever
The temporal model defines the value Forever as a special valid time instant.

RTM8 Interval
Valid time intervals are the chosen timestamp representation.

RTM10 Absolute
All temporal primitives defined are anchored values on the time line.

RMD5 Metamodel of Time and Temporal Concepts A metamodel for time and temporal concepts has been defined.

We mentioned the interval operators as special for intervals, the VTInterval itself is an abstract concept in COMDEF and cannot have operations. The interval operators are therefore not explicitly defined for the VTInterval.

Chapter 6

The Temporal Data Model

The main topic of this chapter is the definition of a temporal data model. The data model is defined by extending the COMDEF framework to form a new UML profile called COMDEF Temporal. Since the extensions rely heavily on the COMDEF metamodel, a closer description of the COMDEF metamodel and a brief example of its usage are provided for the reader in section 6.1. In chapter 3 we recognized the lack of database concepts. Section 6.2 is devoted to the introduction of database specific properties to COMDEF which represents the COMDEF DB profile. The temporal data model is based on COMDEF DB and is described in section 6.3. At the end we present the temporal modeling requirements from chapter 4 which are fulfilled in this chapter.

6.1 The COMDEF Metamodel

The intention with the COMDEF framework was to develop component based distributed systems using a model driven development process [66]. As the framework has a UML profile and strong facilities for mapping models to systems, it is also suitable for general conceptual modeling. However, the framework architecture was not expressive enough to capture the domain specific features we wanted to define. Extensions to the framework was necessary in order to introduce the required concepts, such as database and temporal constructs. In chapter 2 we introduced COMDEF and described the main parts of the framework. In the following, the metamodel used to model the concepts of the architecture is described in more detail. Since the first version of COMDEF, UML has gone through several revisions. One topic for revision has been the identification of several problems with the extension mechanism [39]. Changes and results of the revisions regarding the extension mechanism of UML is not discussed here, we only describe the COMDEF metamodel and its relation to UML as it was first defined.

The COMDEF is a UML profile which extends the UML metamodel.

COMDEF concepts are introduced as new metaclasses in the UML metamodel using the stereotype extension mechanism. Each new metaclass corresponding to a COMDEF concept is defined according to a base class in the UML metamodel. The new metaclass adopts the characteristics of its base class, but will necessarily have specialized structure and behavior. Specialization of the new metaclasses must be specified using OCL to constrain the model elements instantiated from the metaclass. Hence, modeling with the COMDEF, instances of the metaclasses are instantiated as stereotyped model elements.

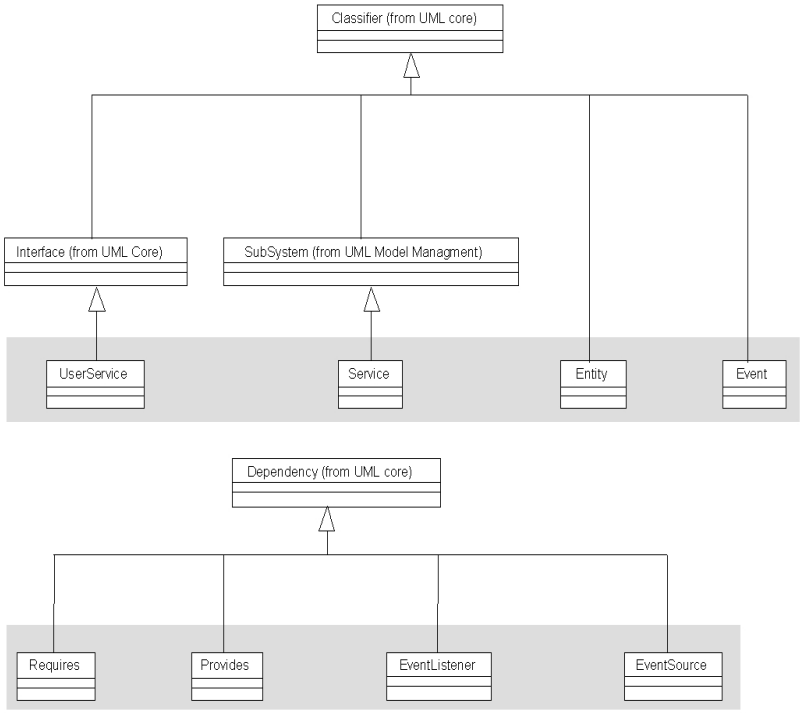


Figure 6.1: Comdef MetaModel

Figure 6.1 shows the COMDEF metamodel/UML profile, and how the new metaclasses relate to the UML metamodel. New metaclasses introduced by COMDEF are the ones within the shaded gray area. As indicated by the figure the metamodel contains two parts. The top model describes the metaclasses that inherit from Classifier or subclasses of Classifier in UML core and are instantiated as stereotyped classes. The bottom model illustrates metaclasses that are extended from Dependency in UML core, these metaclasses differ from the former in that they are instantiated as stereotyped dependencies. Semantics of the dependency metaclasses are not important

here, an example below will point to direction of use.

The concepts in COMDEF was divided into three areas separating the concerns of a distributed information system.

- **UserService** Represents the client side interface to the system.
- **Service** Represents the server side distributed object.
- **Entity** Represents the server side encapsulation of persistent storage, i.e. used to model the information model.

Event is not mentioned above, but represents the event handling between Service and Entity components. Figure 6.2 illustrates the use of the COMDEF UML profile. The example is part of a developed case implementation from the OBOE project [47]. Main features of the COMDEF metamodel are illustrated, where the above described concepts appear in the model as stereotyped model elements. COMDEF uses the UML association model to represent relationships. Constraints for the associations are set to realize the desired COMDEF relationships. The example model shows a report

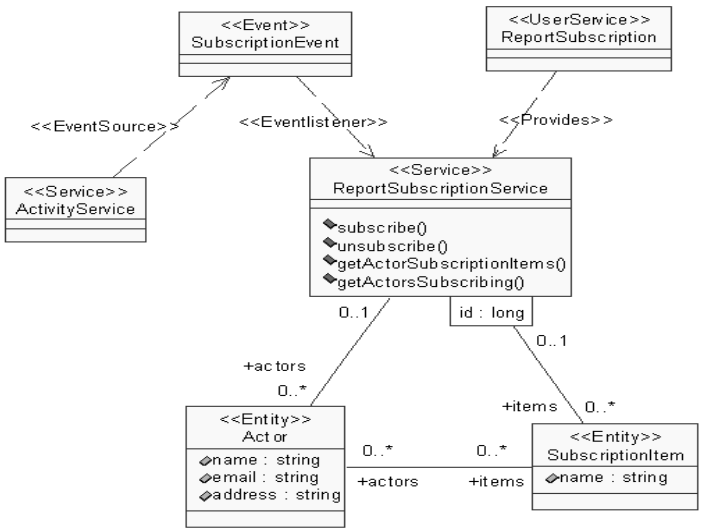


Figure 6.2: A COMDEF metamodel instance

subscription system.

The focus of our work is the data elements being modeled in a system, which in this context is the Entity concept. The distributed part, i.e. Service and UserService, is not within the scope of this thesis. We therefore continue with a closer description of the Entity.

6.1.1 Entity

The concept of an Entity is used to model the information model, that is, entities defined according to Entity are representations of the information objects of interest, e.g. stored in a database. Entity inherits from classifier in UML CORE as can be seen in figure 6.1. Classifier is an abstract superclass for class, interface and datatypes in the UML metamodel [50]. A class diagram stereotyped with the Entity stereotype obtains the characteristics of Entity. An Entity instance has independent existence, is defined to be persistent and is characterized by a set of attributes and a set of methods. Figure 6.3 shows the metamodel which defines the properties of Entity. Similar metamodels of existing and new concepts are used throughout the chapter. These metamodels serve as a basis for describing and understanding the meaning of the modeling concepts and constructs.

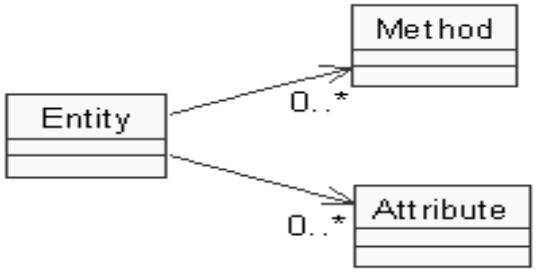


Figure 6.3: Entity metamodel

Relationships are allowed between Entity instances. As mentioned above, the UML association model is utilized. Use of the UML associations are restricted by OCL constraints and the following constraint is imposed at metamodel level using OCL for Entity.

```
context Entity INV associations
self.allOppositeAssocationEnds→ forall (a |
                a.type.OclIsTypeOf(Entity))
```

Entities stereotyped Entity are only simple structures representing information objects without any further semantic. This works well for objects without complex structure, but when modeling more advanced systems the Entity concept is not adequate. Extensions to the Entity concept are therefore the main topic of the forthcoming sections.

6.2 Database Extension

Although the Entity concept is sufficient for modeling e.g. simple static business objects or similar applications without further complexity, certain shortcomings were discovered when trying to use COMDEF to model information models with different and more advanced features, such as database specific properties. A new concept had to be introduced to represent characteristics of database entities and domain specific types more accurately.

The solution was to extend the Entity concept to become a relational database entity, obtaining the properties common to a relational database object. Introduced as a generalization of Entity, we named the database version of Entity RDBEntity. The RDBEntity inherits all features from Entity besides introducing new database specific characteristics. The inheritance hierarchy between Entity and RDBEntity is described in figure 6.4. As can

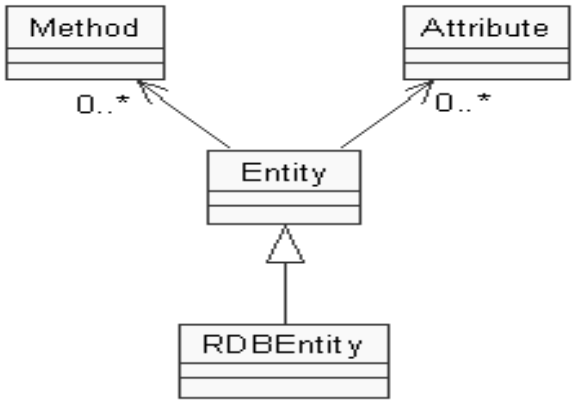


Figure 6.4: RDBEntity inherits from Entity.

be seen from the generalization arrow, RDBEntity is defined to be a subtype of Entity and inherits the structural characteristics of attributes and methods from Entity. We now move on to describe the properties of RDBEntity itself. The model in figure 6.5 shows the metamodel for RDBEntity, which defines the COMDEF DB profile and describes the characteristics of RDBEntity stereotyped entities. Features inherited from Entity are expanded and shown together with the new constructs. RDBEntity strengthens the notion of persistence compared to Entity, for which the only notion of persistence was stated using natural language. As illustrated in the metamodel figure, two new constructs are associated with the RDBEntity, a PrimaryKey and an Index. The meaning and use of these are described in their respective sections below.

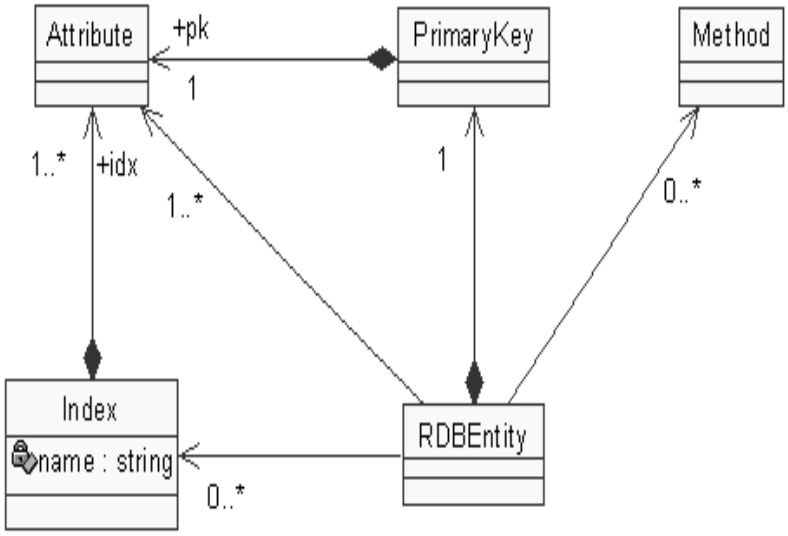


Figure 6.5: RDBEntity metamodel

Primarykey

Depicted in the RDBEntity metamodel the RDBEntity has an associated Primarykey, shown by a composite association from RDBEntity to Primarykey. By the composite association and a multiplicity of one, an RDBEntity defines a single Primarykey by definition. Primarykey again defines *pk*, which references a single Attribute denoted by the composite association to Attribute with role name *pk*. The multiplicity of *pk* is set to one for simplicity, a restriction emphasized by the invariant *PrimaryKey* stated in OCL below. Summing up, RDBEntity defines a single Primarykey which has an associated *pk* referencing a single Attribute. Since *pk* references an Attribute, RDBEntity must at least define one Attribute, expressed by a multiplicity of one or more on the association between RDBEntity and Attribute. Actually, a dependency relationship exists between the mentioned association and *pk*, but no notion of such visual constraints are found in UML.

For a modeled entity stereotyped RDBEntity, i.e an rdb entity, the above characteristics apply. Semantics of the primarykey for an rdb entity are that the selected attribute are to serve as a unique identifier. The following OCL expression states that each RDBEntity entity is required to have a primarykey defined and the attribute referenced by the primary key must exist.

```
context RDBEntity INV PrimaryKey
```

```
self.allinstances->forall( r | select q (r.allattributes()) |
                q = r.primarykey.pk and
                r.allattributes->forall(p | p = q or
                ( p <> q and p <> r.primarykey.pk)))
```

Note, the OCL expression `allattributes()` returns all attributes of an entity, both local and inherited. We have defined the expression in order to simplify the OCL expression and thereby the readability. A primary key cannot be redefined through inheritance. All entities with an RDBEntity stereotyped parent inherits the primarykey definition of its parent, and is maintained by the following OCL expression of the RDBEntity.

```
context RDBEntity PrimaryKeyInherited
self.allinstances-> forall( r | r.allSupertypes->forall(
                s | r.primarykey.pk = s.primarykey.pk))
```

Index

The database entities can define indexes. Indexes are used to optimize retrieval from persistent storage. The metamodel in figure 6.5 shows Index as an association from RDBEntity. By the multiplicity we see that zero or more Index definitions are possible. Index itself has a name and an association named *idx* which references Attribute. The association *idx* referencing Attribute have a zero or more multiplicity, that is, the *idx* can reference zero or more elements of Attribute. To be perfectly precise, a dependency restriction is existent between the associations *idx* to Attribute and from RDBEntity to Attribute. No index can have a reference to an attribute that is not defined for an RDBEntity instance. Similar as for the primary key above, a restriction hard to express visually using UML.

General Constraints on RDBEntity

A general constraint inherited from UML Core is that only leaf nodes of RDBEntity instances are instantiable, that is, abstract instances are not implemented.

The next OCL expression constrains the relationships of RDBEntity. Database entities are used to model the information model only, and relationships to other kinds of model elements are therefore not allowed. Below we state that rdb entities are only allowed to have relationships to other rdb entities.

```
context RDBEntity INV associations
self.allOppositeAssocationEnds -> forall (a |
                a.type.OclIsTypeOf(RDBEntity))
```

6.3 The Temporal Profile

This section presents the temporal extension to the framework and the semantics and usage of the concepts in the Temporal profile. A profile is a set of extensions to referencing UML profiles, these extensions take the form of metamodels representing a specific domain. The former section described the COMDEF database profile which the temporal profile is based on.

Main features of the temporal profile is introduced in the next section. We describe the temporal metamodel which defines the temporal data model. Requirements for such models was defined in chapter 4, requirements fulfilled by the temporal data model defined below are described at the end of this chapter. We have striven to define a simple and general temporal data model having the essential features to capture the time varying nature of information. A combination of precise metamodeling using UML and refinement of extended model elements using OCL are used to achieve this goal. A general requirement for the Temporal profile is that it should be a proper UML metamodel extension conforming to the restrictions set by UML.

6.3.1 Temporal Metamodel

In order to extend the existing framework with temporal concepts, a few different approaches where considered. Using UML to represent domain specific aspects is not straightforward. Introducing temporal concepts to existing models are no exception, as mentioned in section 2.2.4 a variety of different approaches are possible. Above we extended COMDEF with database specific properties to form the database profile. The COMDEF DB defines a snapshot data model, that is, the RDBEntity supports only current states of entities. Extensions to the COMDEF DB were therefore necessary to support the time varying nature of entities. We extend the COMDEF DB to define a temporal data model where the temporal support is defined within the data model. That is, according to the approaches described in chapter 2 we use the extension approach to define the temporal data model. This led to the introduction of a new concept orthogonal to the RDBEntity, namely the ValidTimeEntity. Separation of concerns are important for conceptual models, in general, orthogonality is the key to solve dimensionality [68].

The ValidTimeEntity

Temporal constructs are introduced to the COMDEF framework by the new concept ValidTimeEntity. Our aim is to model temporal variation of real world objects using the relational model as the underlying architecture. Thus, database specific characteristics are equally important for the

ValidTimeEntity as for the RDBEntity. The natural solution was to let the ValidTimeEntity inherit all characteristics of the RDBEntity. In figure 6.6 the ValidTimeEntity concept is defined as a subtype of the RDBEntity. Hence, the ValidTimeEntity inherits all characteristics defined in the RD-

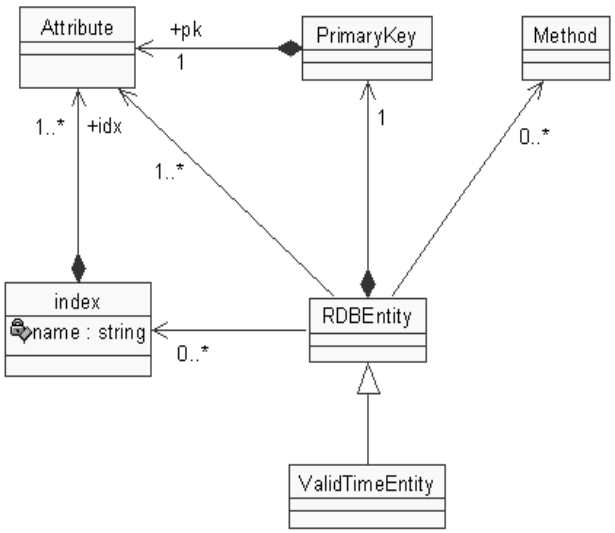


Figure 6.6: ValidTimeEntity as subtype of RDBEntity

BEntity. This implies that the ValidTimeEntity defines a set of attributes, a set of methods, a single primarykey definition and a set of indexes. The semantics of these are described in section 6.2. Figure 6.7 describes the inherited characteristics of the ValidTimeEntity, note the implicit generalization relationship to the RDBEntity.

Timestamping the ValidTimeEntity

In order to capture the time varying properties of an entity, it must have an associated timestamp. Past, present and future states of a real world entity are to be captured by the ValidTimeEntity. The level of timestamping may take the form of entity timestamping, associating a timestamp to the entity as a whole, or timestamping each attribute of an entity separately. We chose the approach of timestamping complete entities. The decision was based on the underlying architecture, attribute timestamping using the relational model is technical and difficult to implement. A requirement was to provide simple platform independent support for modeling temporal entities for persistence. Although we model real world entities in UML, that is, in an object model, by extending the RDBEntity our main target is the

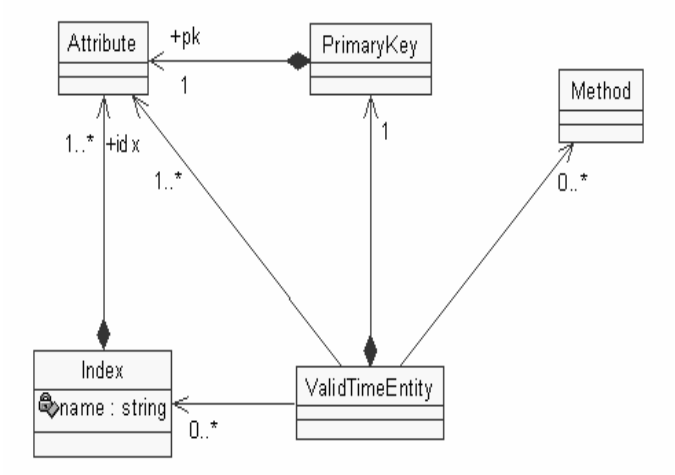


Figure 6.7: Inherited Characteristics of ValidTimeEntity

relational model as underlying data model. Entity timestamping was chosen as a simple and viable solution when using the relational model.

Entity timestamping associates a value of time with each entity, in our case valid time was the chosen aspect of temporal support, hence facts represented by an entity are timestamped by values from the valid time dimension. The timestamping of the ValidTimeEntity is realized by the following. The ValidTimeEntity is associated with a new timestamp construct. The new construct of the temporal entity is, as shown in figure 6.8, constructed of a ValidTimeEntity and its associated *VTTimestamp* referencing a VTInterval.

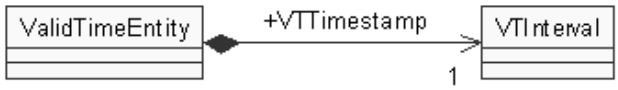


Figure 6.8: ValidTimeEntity with VTTimestamp referencing VTInterval

In the figure the association is composite and of single multiplicity, semantics is that ValidTimeEntity has by definition one and only one associated *VTTimestamp* of type VTInterval. VTInterval was defined in chapter 5 as a valid time interval. To get an overview of the different parts we introduce the temporal concepts again, this time together with the ValidTimeEntity to complete the model. In figure 6.9 a complete view of the ValidTimeEntity and the associated time and temporal concepts are shown. In the figure all characteristics of the ValidTimeEntity are described. A ValidTimeEntity has an associated *VTTimestamp*, the *VTTimestamp* ref-

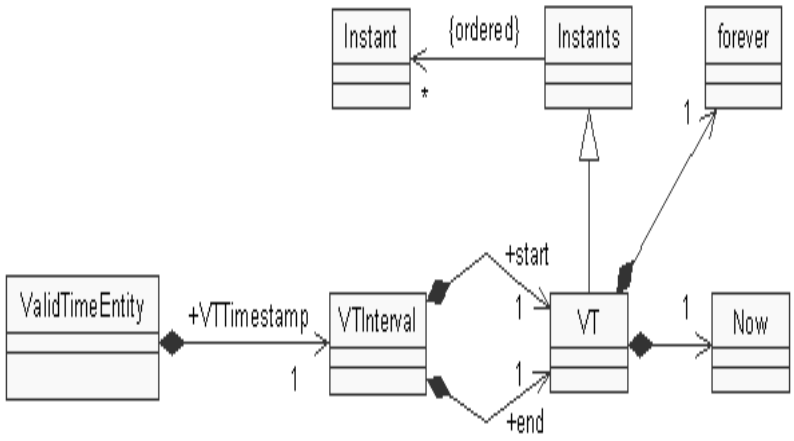


Figure 6.9: Relationship between ValidTimeEntity and the temporal model

erences a VTInterval. VTInterval is a valid time interval representing the time between two valid time instants from the valid time domain VT. This means that the ValidTimeEntity has an associated set of valid time instants from the valid time dimension.

All modeled entities stereotyped ValidTimeEntity adopts the characteristics defined for ValidTimeEntity above. When classifying an entity with the ValidTimeEntity stereotype, the valid time constructs are instantiated. Semantics denoted by *VTTimestamp* is interpreted as the duration of time when facts, that is, the different states of an entity, were valid in the modeled world. A closer description of the semantics of the ValidTimeEntity is found in section 6.3.2.

Constraints on ValidTimeEntity

Behavior of valid time entities must be constrained, the next OCL expressions define restrictions on the ValidTimeEntity. A valid time entity do have a timestamp. The OCL expression below states the required definition of a *VTTimestamp* of type VTInterval for each valid time entity.

```
context ValidTimeEntity INV VTTimestampDefined
self.allInstances -> forall(r |
               exist (r.vttimestamp) and
               r.vttimestamp.OclIsTypeOf(vtinterval))
```

Primary keys serve as unique identifiers. Because the states of entities vary in time, objects having identical keys will exist. The following OCL expres-

sion states that two instances of a valid time entity having identical primary keys cannot have identical *VTTimestamp* values.

```
context ValidTimeEntity INV timeinvariantprimarykey
self.allInstances -> forall ( E | E.allInstances->
               forall(e1,e2 |
               e1.primarykey.pk = e2.primarykey.pk implies
               e1.vttimestamp <> e2.vttimestamp))
```

From the definition of VTInterval the constraint below is defined, but not in the context of ValidTimeEntity. Each timestamp associated with a ValidTimeEntity must be a valid VTInterval.

```
context ValidTimeEntity INV VTTimeStampCorrect
self.allInstances -> forall (E | E.allinstances->
               forall( e | e.vttimestamp.start < e.vttimestamp.end))
```

COMDEF Temporal utilize the relationship model originally defined in COMDEF, the model and issues regarding temporal relationships are described in more detail in section 6.3.3. The following OCL expression states that entities stereotyped ValidTimeEntity are only allowed to have relationships to other temporal or database entities.

```
context ValidTimeEntity INV associations
self.allOppositeAssocationEnds -> forall (a |
               a.type.OclIsTypeOf(ValidTimeEntity) or
               a.type.OclIsTypeOf(RDBEntity))
```

Operators

The interval comparison operators described in section 5.2.1 are now defined. In figure 6.10 all characteristics of the ValidTimeEntity are illustrated, the implicit interval operators can be viewed as operations of the entity. Note, the operators are defined for operations on the valid time intervals only, and have no other function than comparing valid time intervals of temporal entity states.

ValidTimeEntity introduces the comparison operators as implicit binary operations for a temporal entity. Each operator returns a boolean deciding the result.

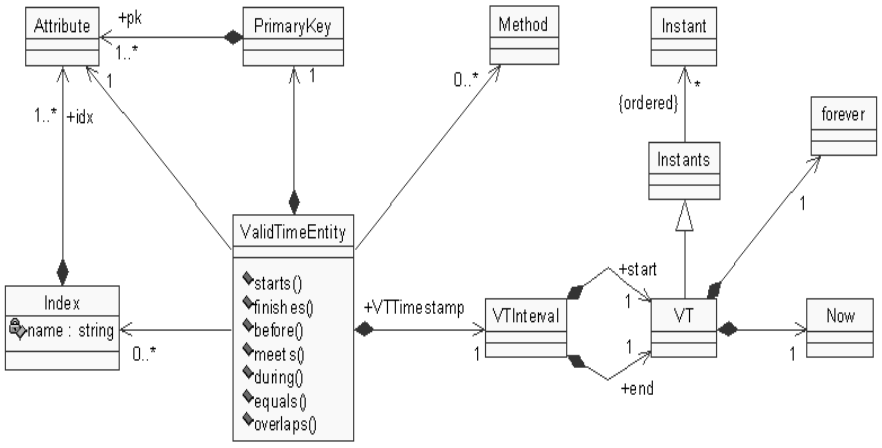


Figure 6.10: All characteristics of ValidTimeEntity

6.3.2 Properties of Valid Time Entities

Above, abstract syntax and notation for the valid time entities was described. Semantics are hard to express using UML, and other mechanisms provide a more natural explanation of the time varying entity. The disadvantage of using an entity timestamping approach is that a real world entity is represented by several tuples in the database. A representation awkward for an entity as a single unit. Another is the issue of redundancy, frequent changes in single attributes of an entity lead to a high level of data repetition. Below we elaborate on some issues regarding the valid time entity.

Let $\mathcal{E}$ be a modeled entity defined according to ValidTimeEntity, that is, $\mathcal{E}$ is stereotyped with the ValidTimeEntity stereotype. Then $\mathcal{E} = \{E_1, \ldots, E_n\}$, $1 \leq n$, which defines $\mathcal{E}$ as a set of valid time entity instances, i.e., each entity instance is a representation of a real world object. As real world objects vary in time, so do the entity instances and the information history regarding an entity is composed of a set of states. For an instance $E_i \in \mathcal{E}$, $1 \leq i \leq n$, the set of states is denoted by $E = \{e_1, \ldots, e_n\}$,$1 \leq n$. Each $e_i \in E$, $1 \leq i \leq n$, contains a set of values representing the facts of the respective state of E.

One value acts as the unique identifier of E, for each $e_i \in E$, $1 \leq i \leq n$, this value is denoted by $e_i.primarykey$. Moreover, each $e \in E$ has an associated valid time timestamp $VTInterval \in I(VT)$, where $I(VT)$ is the set of all valid time intervals as defined in section 5.2.1.

Lifespan of an entity is a concept not discussed, as we do not explicitly capture lifespan of entities. The lifespan encompasses the valid time of any state, i.e., data instance of the entity object. Some data models explicitly support lifespan of entities, in our model lifespan of an entity is implicit and

deduced by computing the union of valid time intervals for each state of the entity.

Since the set E represents states of the entity instance, the states will have identical primary key values. This implies that further restrictions on the set E have to be defined. Each entity instance in the set must be unique, which is enforced by stating that each primary key is time-invariant. The primary key constraint on states of valid time entity instances set that none of the valid time timestamps of entities with identical primary keys can be equal. The constraint is defined as follows.

Definition 6.3.1 ***time-invariant primarykey constraint***
$\forall e_i, e_j \in E\ (e_i.primarykey = e_j.primarykey \Rightarrow$
$e_i.vttimestamp \neq e_j.vttimestamp)$

The definition is important for satisfying the set restriction and fulfill the responsibility of the primary key as a unique identifier. The primary key is the only time-invariant attribute in a valid time entity instance, all other attributes are allowed to vary in time. This constraint is also expressed in OCL above.

The following is an example valid time entity *Property* defined in CML.

```
validtimeentity Property  {
      attribute string id;
      attribute string tenant;
      vttimestamp vtinterval;
      primarykey id;
};
```

Table 6.3.2 is an example of entity instances in a database of the valid time entity Property shown in CML above. Let us imagine that a real estate company keeps a record of the properties they manage and their tenant's history over time. Sam hired property 12 from 1982 until 1991, represented at that time by an insertion of an entity E with the fact e =(12, Sam | 1982 -*now*). This was the first registered tenant for house number 12 and thereby the first time the house was instantiated as an entity in the system. In 1991, the property was scheduled for renovation and his period as a tenant was ended by setting the vttimestamp.End timestamp to 1991. In 1993, Sam was registered as tenant for property twelve again. From the vttimestamp.End value of *Now* in the second tuple we can see that Sam still is the current tenant of property twelve. We can also deduce that Liv, Sam's girlfriend, is also a registered tenant of the house from 1997. In 1994, Paul hired property number 23 with a contract lasting 5 years, as the complete tenancy period was known at the moment of registration, the vttimestamp was set to [1994,1999]. When Paul's contract ended, he decided to renew

id	tenant	vttimestamp
12	Sam	1982 - 1991
12	Sam	1993 - now
12	Liv	1997 - now
23	Paul	1994 - 1999
23	Paul	2000 - now

Table 6.1: A table showing tenant history

the contract and rent the property for an unknown time. This resulted in an insertion of a new entity stating that Paul rents property 23 from 2000 until the current time.

If we look closer at the table 6.3.2, a few interesting properties of the set of entities can be discussed. Remember that the attribute values of a state denote a fact, and the timestamps define the valid time of the state.

A concept commonly used in temporal database terminology is value-equivalence.

Definition 6.3.2 ***Value Equivalence*** *[30] Two entity states $e_i, e_j \in E$ are value equivalent if they have identical (non-timestamp) attributes values.*

In our example the two entity states associated with property 23 is value equivalent. Value equivalence is usually a term used in conjunction with the concept of coalescing. Coalescing is an operation that, in this case, merges value equivalent entity states with consecutive valid time intervals, thereby replacing two states with a single state having an enlarged valid time interval [7]. Paul has two periods of tenancy which are adjacent in time and could be coalesced. In this case, coalescing will be space preserving, but we will lose the information that Paul once renewed his contract. No coalesce operation is defined in our model, such an operation defines point based behavior of a model.

Whether a data model is point or interval based is often found to be confusing. The question is how the intervals are preserved, when timestamping facts with intervals the intervals only serve as a shorthand notation for a set of instants. Informally a data model is defined to be interval based if all facts are interval timestamped over the time point domain, and all the operators in the model are interval based, i.e. the intervals are preserved during the operation. The same applies for a point based model, only vice-versa [5]. Our model is defined as interval based, the intervals are respected in all operations defined as of now. Each valid time interval is treated as an atomic value.

6.3.3 Relationships in COMDEF

Modeling the relationships between entities are important in an information system. COMDEF has a relationship model similar to that of the notion defined by the ODMG [9]. By utilizing the already existing association model defined in the UML metamodel and defining some behavioral rules, the desired COMDEF relationships such as single and list relationships are defined. Bag, set and dictionary relationships are also supported.

Managing relationships between entities with valid time semantics is somewhat more problematic. A general assumption is that relationships between valid time entities are valid only during the intersection of the lifespan of the participating entities. A similar problem is present regarding relationships between general database entities and temporal entities. Another issue is the intended meaning of the relationship, situations may occur where the above assumption is inappropriate. One option is to design a temporal extension to the relationship model in COMDEF, but this was not investigated closer due to technical complexity and time constraints. The COMDEF DB and Temporal profiles utilize the relationship model as it is. Management of relationships involving temporal entities are left to the application.

6.4 Data Type Extension

COMDEF supports only Corba IDL atomic data type definitions. Based on the CORBA IDL type hierarchy, models mapped to different architectures utilize the IDL mappings only. In order to use COMDEF for different domains we need to allow definitions of domain specific data types. New data types could have been added by extension to the CML grammar, but this would violate the original design and misuse might have caused inconsistencies in existing COMDEF models. COMDEF as a model driven framework should be language neutral and domain specific types should therefore not be a static part of the architecture.

The solution was to extend the framework to support inclusion of externally implemented data types. Domain specific data types not recognized by the original framework can then be included in models. Extension was done by introducing a new *Datatype* concept to COMDEF. The metamodel is quite simple, a *Datatype* is a sort of classifier and its only characteristic is a set of possible operations. In figure 6.11 the definition of the *Datatype* in UML is represented. Instances of data types are primitive values or objects without identity. All operations owned by a data type must be queries, i.e. they are pure functions and can only return values, not modify values. The constraint is maintained by the following OCL expression.

```
context Datatype INV OperationsAreQueries
```

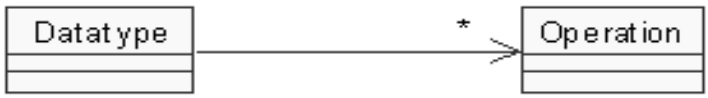


Figure 6.11: The data type extension

```
self.ownedOperations->forall(f | f.isQuery)
```

The COMDEF DB and COMDEF Temporal profiles have a "uses" relationship to the external type library. The new data types must of course be recognized by their respective domain specific mappings, that is, the implementation of the types are external to COMDEF. Since the data types are external and only represent an interface to the implemented types, associations and attributes of Datatype are not supported.

Two data type libraries are defined for the COMDEF DB and COMDEF Temporal profiles. A time related data type library and a spatial data type library is available.

6.4.1 Temporal Types

User defined time is supported by a set of time related data types in COMDEF. The data type library of temporal types in COMDEF Temporal is depicted in figure 6.12. Datetimes are a common notion for data types representing dates and times. Datetimes included in the model are in conformance with the SQL-92 standard as defined in [43]. Semantics of the

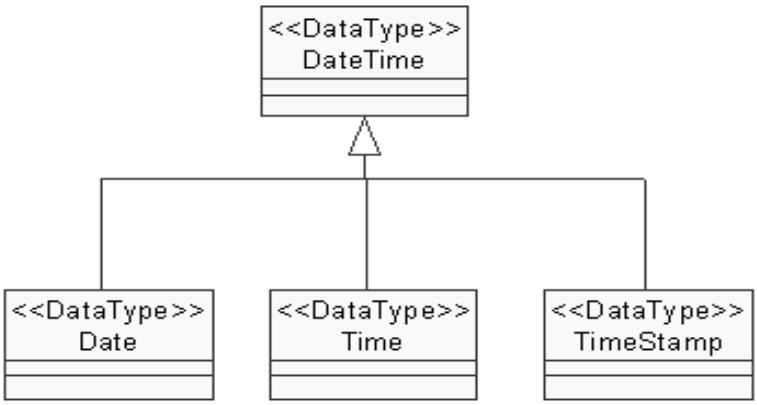


Figure 6.12: Time related data type library

included datetimes are defined below.

Date	Stores year, month and day of a point on the time line. E.g. 25/11/2001.
Time	Stores hour, minute and second. Can be used to store recurring points of time, or a point on the time line where date is implicit. E.g. 13:20:53.
TimeStamp	Stores year, month, day, hour, minute and second of a point on the time line.

6.4.2 Spatial Types

The spatial type library defines the available spatial types. Work on spatial types in the COMDEF framework has been done in [81] and is not elaborated over herein. In figure 6.13 the spatial types are described using UML. The

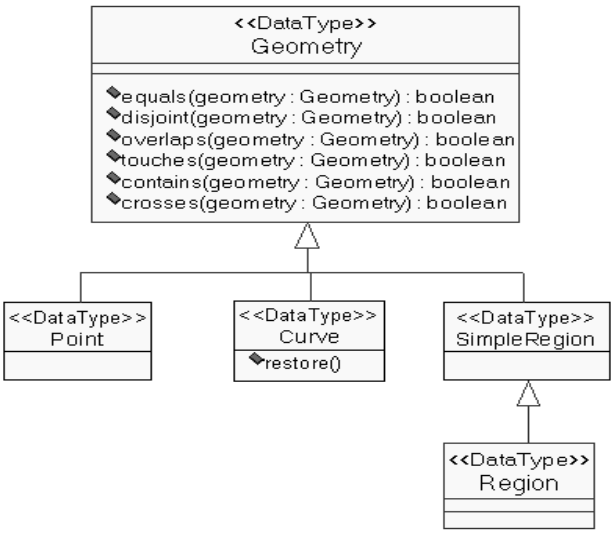


Figure 6.13: Spatial type library

supertype *Geometry* defines binary spatial operations for the spatial extents.

6.5 CML Grammar Extensions

So far the lexical language CML (Component Modeling Language) has received little concern. The intention with the lexical language is to provide a textual notation complete with respect to the graphical representation

of models. Mismatch between graphical notation and lexical notation is a general problem in software development frameworks. CML is based on extensions to the existing lexical notations of IDL (Interface Description Language) [48] and ODL (Object Definition Language) [9]. To provide a textual representation of COMDEF models the concepts in the profile are defined in CML. The next sections present the grammar additions to CML used to define the DB and Temporal profile concepts. As the grammar is not a key topic, the descriptions are brief.

RDBEntity

CML grammar for RDBEntity is defined in figure 6.14. Two main grammar productions are added compared to the Entity grammar; `primarykey_dcl` and `index_dcl`. Characteristics of Entity are obtained by the grammar

rdbentity_dcl	::=	<rdbentity> indentifier (*inheritance_spec*)? (“ (“ rdbentitybody_dcl “) “)?
rdbentitybody_dcl	::=	(*export_rdbentity*)∗
export_rdbentity	::=	export_entity | primarykey_dcl “;“ | index_dcl “;“
primarykey_dcl	::=	<PRIMARYKEY> key_dcls
index_dcl	::=	<INDEX> key_dcls
key_dcls	::=	key_dcl (“,“ key_dcl)*
key_dcl	::=	identifier

Figure 6.14: RDBEntity CML grammar

production `export_entity`.

ValidTimeEntity

Figure 6.15 shows the CML grammar definition of ValidTimeEntity. The grammar production `timestamp_dcl` defines the timestamp and `vttype_dcl` states the timestamp type. All other characteristics of ValidTimeEntity are obtained from RDBEntity by the production rule `export_rdbentity`.

Datatype

The CML grammar definition of the Datatype extension is depicted in figure 6.16. A data type consists of a *Datatype* declaration and an identifier, the grammar production `datatypebody_dcl` defines one or more operation declarations, that is the `op_dcl`.

validtimeentity_dcl	::=	<VALIDTIMEENTITY> indentifier (*inheritance_spec*)? (“ (“ validtimeentitybody_dcl “) “)?
validtimeentitybody_dcl	::=	(*export_validtimeentity*)∗
export_validtimeentity	::=	export_rdbentity | timestamp_dcl “;“
timestamp_dcl	::=	< VTTimestamp > vttype_dcl
vttype_dcl	::=	<VTInterval>

Figure 6.15: ValidTimeEntity CML grammar

datatype_dcl	::=	<DATATYPE> identifier (*inheritance_spec*)? (“ (“ datatypebody_dcl “) “)?
datatypebody_dcl	::=	(*export_datatype*)∗
export_datatype	::=	op_dcl “;“

Figure 6.16: Datatype CML grammar

6.6 COMDEF Profile Architecture

The Comdef DB and Temporal profile have been defined above. We now present an overview of the relationships between the profiles, a typical scenery for a complete use of COMDEF. Illustrated in figure 6.17, the COMDEF profiles are described using packages stereotyped with the profile stereotype. Packages in this context are synonyms for profiles and serve as a mechanism useful for structuring profiles. In the figure, the arrows denote dependency or uses relationships between packages.

The DB profile uses the COMDEF profile and the Temporal profile uses the DB profile. Data type libraries are available both to the DB and Temporal profiles, for readability the DB profile relationships to the data type libraries are left out. Two packages are stereotyped DataTypeLibrary, the TemporalTypes and the SpatialTypes. The COMDEF DB and Temporal employs the types defined in the data type packages as an included data type library.

Two model packages are included in the figure. They are used to illustrate the use of the different COMDEF profiles. A model designed using one of the profiles is modeled in a module, stereotyped with the Module type in the figure. Two different modules are shown, the DBSchema module and the Application module. Models using the Temporal profile constitute the persistent system information model and are modeled in the DBSchema package. Applications using the information model are designed in the Application package, these models may utilize the original COMDEF stereotyped concepts of Service and UserService. A GUI profile, not described in the figure, has also been developed, which provides modeling

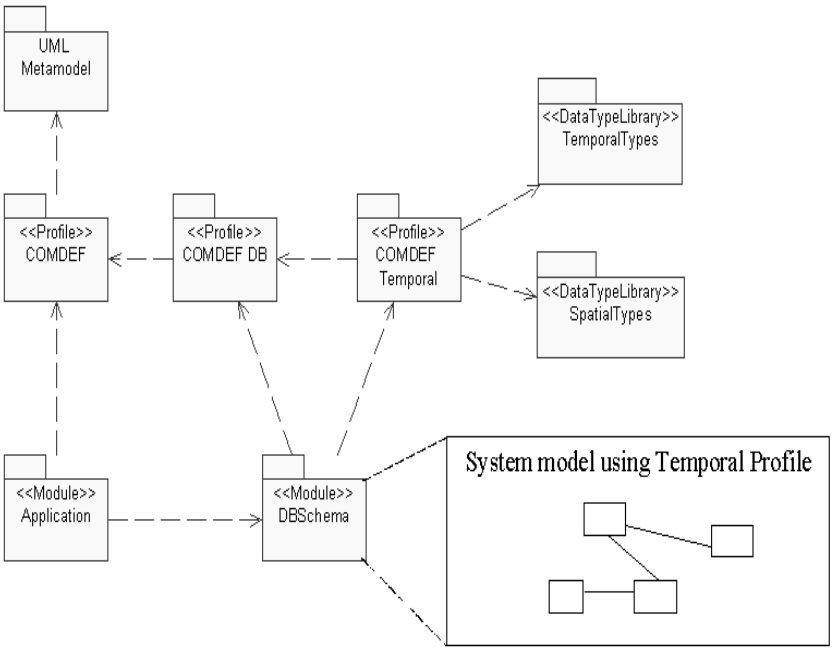


Figure 6.17: Complete COMDEF Temporal profile architecture

concepts for the purpose of designing graphical user interfaces based on the application model.

6.7 Summary

In this chapter we have presented a temporal UML profile. The COMDEF Temporal defines a temporal data model based on timestamping entities. The UML profile provide modelers with a simple and expressive modeling language sufficient to design information models having valid time support. A data type extension to COMDEF have been described, allowing the framework to be extended with data types for a specific domain.

The temporal data model defined fulfills the following requirements for temporal data models and model driven development defined in chapter 4.

RTM3 User Defined Time
User defined time is supported by the inclusion of a set of time related data types.

RTM12 Interval Operators
Allen's comparison operators are defined for each entity to compare intervals.

RDM2 Timestamped Entities
A valid time timestamp of type valid time interval is associated with each temporal entity.

RDM5 Time-invariant Identifiers
Primary keys serve as unique identifiers, constraints on valid time entities ensure that the key is time-invariant.

RMD1 Methodology Support
The temporal data model is defined using the UML and methodology support is ensured.

RMD2 UML Profile
The COMDEF Temporal is a proper extension to UML and thus a UML profile.

RMD3 Interpretable Mapping from Graphical Notation
Models are mapped via the XMI. Since COMDEF Temporal is a UML profile and thus conforms to the UML metamodel, models can be interchanged using the XMI format. In section 7.4 the mapping process in COMDEF is described closer.

RMD4 Platform Independent
The COMDEF Temporal is language neutral and can be mapped to any underlying architecture. Temporal and database concepts are profile specific specializations only.

RMD6 Simple and Expressive
The COMDEF Temporal is a simple temporal model, concepts provided are few and easy to comprehend. The model provides a clean view of the modeled elements and is at the same time sufficiently expressive.

An example of usage of the temporal profile is described in chapter 7 where the profile is used to implement a case application.

Chapter 7

Case Revisited

In this chapter the land information system model from chapter 3 is reintroduced, now using COMDEF Temporal. We describe the usage of the new modeling constructs and their properties. Further we present an implemented case application based on a mapping from the model.

7.1 The System Model

Figure 7.1 shows a model of the LIS system modeled using the COMDEF Temporal profile. All new concepts are included in the model. In the following we describe the concepts and relate them to the requirements presented in the case model in chapter 3. When modeling with COMDEF, the specific concepts and constructs are instantiated by using the COMDEF Temporal profile. The profile also instantiates the data type library, which gives the modeler access to a set of temporal and spatial data types. For demonstration purposes all entities are stereotyped by the ValidTimeEntity stereotype, i.e. all entities are temporal. COMDEF is flexible; both temporal and non temporal entities can be mixed, providing optional temporal support. The model presents a clear and visible view of the system, it retains simplicity and understandability even when the modeled entities have temporal aspects. Unfortunately, the representation of the domain specific constructs cannot be expressed using conventional UML, a topic to be discussed in chapter 7.3. In COMDEF however, the constructs are defined and we will describe the definitions as we discuss the new concepts.

7.1.1 Database Support

In the original case model we had no notation to define persistent entities and their properties. The ValidTimeEntity stereotype introduces persistence to the model elements. By definition all valid time entities are relational database entities. In the case model each entity is stereotyped ValidTimeEntity

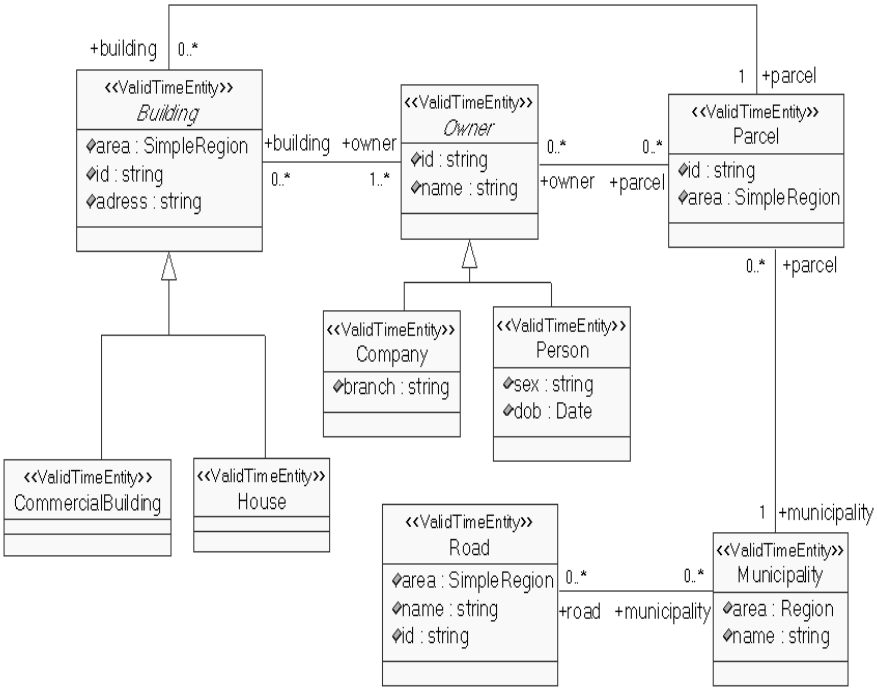


Figure 7.1: A revised case model

which identifies the entity as a persistent database entity.

Primary keys

Each valid time entity has a primary key defined. The primary key is instantiated when stereotyping an entity with the ValidTimeEntity stereotype, an attribute must be designated as the primary key attribute by the modeler. The primary key is defined by *primarykey* keyword. For entity owner the following CML is defined to state *id* as a primarykey.

```
primarykey id;
```

The primary key is defined as inline CML of the entity in a model.

Index

Index is an optional property of database entities. A user is allowed to define an attribute of an entity as an index. Similar to the primary key, indexes cannot be seen in the model itself. To denote *name* as index for

entity owner the following CML must be defined using the *index* keyword in CML.

```
index idx name;
```

One or more attributes of an entity can be indexed, the *idx* denotes the index name.

7.1.2 Valid Time of Entities

All entities in the case model are temporal. The valid time entity allows us to model entities that vary in valid time, i.e. the states of the entity at different times are captured, denoting when the facts represented by the entity was valid in the modeled reality. When modeling using the temporal profile, the system modeler has to decide if valid time of an entity is to be supported. Entities chosen to be temporal must be classified with the ValidTimeEntity stereotype. The stereotype instantiates the valid time timestamp construct described below.

Valid Time Timestamp

All entities stereotyped by ValidTimeEntity have by definition a valid time timestamp, denoted *vttimestamp* of type *vtinterval*. The timestamp is applied to each entity when an entity is classified with the ValidTimeEntity stereotype. In CML the statement below is mandatory for each valid time entity.

```
vttimestamp vtinterval;
```

7.1.3 Domain Specific Types

The COMDEF temporal profile has a *uses* dependency relationship to the data type libraries. When modeling with the profile, these types are instantiated as if they were regular data types in COMDEF. This allows modelers to use them in a flexible way; both general database entities, i.e. rdb entities, and valid time entities can use the data types freely. This is shown in the model. The attribute *area* which has a type of *Region* or *SimpleRegion*, found on the municipality and the building entity respectively are examples of spatial types. User defined time is present in entity Person; where an attribute date of birth (dob) of the type *Date* is defined. To COMDEF itself, the data types are external. Their implementation is known to the mapping facilities only.

7.2 Example CML Definition from the Model

We show the resulting CML notation of the owner entity and its subtypes to fully describe the meaning of the case model. All features introduced in the previous sections are shown. For brevity, relationship definitions are not included. Also, the module keyword denotes the package structure in COMDEF. All database entities are modeled in a separate package, in this case DBSchema.

```
module DBSchema{
  abstract validtimeentity Owner  {
           attribute string id;
           attribute string name;
           vttimestamp vtinterval;
           index idx name;
           primarykey id;
  };

  validtimeentity Company : DBSchema::Owner {
          attribute string branch;
  };

  validtimeentity Person : DBSchema::Owner {
          attribute string sex;
          attribute Date dob;
  };
}
```

The CML representation and the model map one-to-one, even if the domain specific constructs cannot be depicted in the model diagrams.

7.3 Representing Domain Specific Constructs

Stereotypes served as a functional extension to denote domain specific concepts at entity level, but we discovered problems when trying to express the extensions at property, i.e. attribute level. The question was how the new domain specific constructs should be introduced in a model. This was a result of the singular property of stereotypes as described in section 2.1.3.

In a situation the modeler may want to define an index using the same attribute as the designated primary key attribute. If the primarykey was defined using a stereotype, a new index stereotype cannot be used on top. Note, a primarykey is actually an index, the above is just an example to point out the problem. One solution, which is allowed, is to construct composite stereotypes of other stereotypes, any stereotype may be constructed

as specializations of numerous other stereotypes. A solution which results in a complex hierarchy of artificial stereotypes not desirable. Because we ruled out stereotypes use at attribute level in our research, we had to find another way of representing the domain specific constructs.

The valid time timestamp construct is instantiated by the ValidTimeEntity stereotype. Primary keys and indexes are different in that the attribute for specification is user defined. The primary keys and indexes of COMDEF profiled entities are therefore specified externally to the model element using inline CML. This was only due to our restriction on the use of stereotypes, primary key and indexed attributes can easily be specified by stereotypes.

7.4 From Model to Generated Code

Mapping the case model in COMDEF Temporal to generated code requires a few steps. The following describes the technologies used to accomplish them.

System Model to XMI

In section 2.1.2 we described XMI as a textual MOF based interchange format. XMI export facilities in current modeling tools are used to map the designed model using COMDEF Temporal to XMI. The result is a textual description in XML format of the model. It is a one to one mapping between the model and the XMI generated. By mapping the model via XMI, we avoid tool dependency and provide interchange of models across platforms.

XMI Mapping to CML

The XMI representation of the model is mapped to the lexical language CML using XSLT (Extensible Stylesheet Language Transformations). XSLT is an XML transformation language defined by W3C in 1999 [80]. To be more specific, XSLT transforms an XML document into a different presentation format after an XSL (Extensible Stylesheet Language) stylesheet specification. Redundant information is removed during the transformation, but there is an one to one mapping between the information model in UML and the CML representation.

Code Generation from CML

The code generation facilities are among the strongest features of COMDEF. The CML textual representation is used by the code generation tools in COMDEF to develop the system. A mapping developed from COMDEF Temporal is described briefly in the next section, where generated code based on a subset of the case model is used by an implemented application.

7.5 Implemented Case Application

We decided to create a map application to be viewed in an Internet browser. A client was developed to show a map of a larger region, e.g. a municipality, and have the ability to show time-varying information of thematic values, in this case building owners. A subset of the case model was used to develop the system. The new modeling constructs are utilized by the code generation tools in COMDEF to minimize difficult manual implementation. Minimizing manual implementation, in this case, consist of handling domain specific functionality, e.g. the generation of code to ease the management of temporal data and access code for spatial objects. Together with more simple features as database schema generation, the functionality gives us the capability of efficiently developing system code to be used by a LIS application.

7.5.1 Implemented Application Architecture

The following describes the architecture of the implemented map application. To support a distributed environment Enterprise Java Beans was chosen as an architecture to be used as a wrapping on the generated code. Enterprise Java Beans is a component architecture for development and deployment of component-based distributed applications. EJB provides transactional, secure and scalable services to applications [16]. The EJB architecture is powerful in the sense that neither the bean developer nor the client application programmer needs to be concerned with complicated and error-prone server side issues. EJBs run in a container which provides support for the issues described. In our case the container is the Oracle9i Application Server. Oracle9i Application Server provides a complete Java2 Platform, Enterprise Edition container which includes a servlet engine and an EJB container. Java2 Platform, Enterprise Edition (J2EE) specifies a standard architecture for developing multitier services for enterprises [73].

An illustration of the complete architecture can be seen in figure 7.5.1. We have developed a multitier architecture with loose coupling in both ends of the middletier. The two end tiers consist of a database and a thin client application, while the middletier contains all the information logic. Actually, the figure describes all the application components required in a proper J2EE application.

The Database Tier

The Oracle database is used for persistent storage in the system. The case model is mapped to relational schemas in the database. Oracle provides a spatial module to store geographically referenced data conveniently. The database was populated with owner histories and spatial extents.

The Application Server

The illustration in figure 7.5.1 gives a logical view of the different elements in the application server. Two containers are depicted inside the J2EE container, a web container and an EJB container. Typical web components such as servlets and e.g. Java Server Pages (JSP) execute in the web container. Servlets receive requests from a client and dynamically generate a computed response. A web listener, in this case Oracle HTTP Server Apache, directs client request to servlets. A servlet residing in the web container serves as the first tier in the application server architecture. Second tier is the EJB coupled with a generated Java access layer from COMDEF Temporal and deployed to the EJB container. These components are actually two separate tiers, but are both deployed to the EJB container.

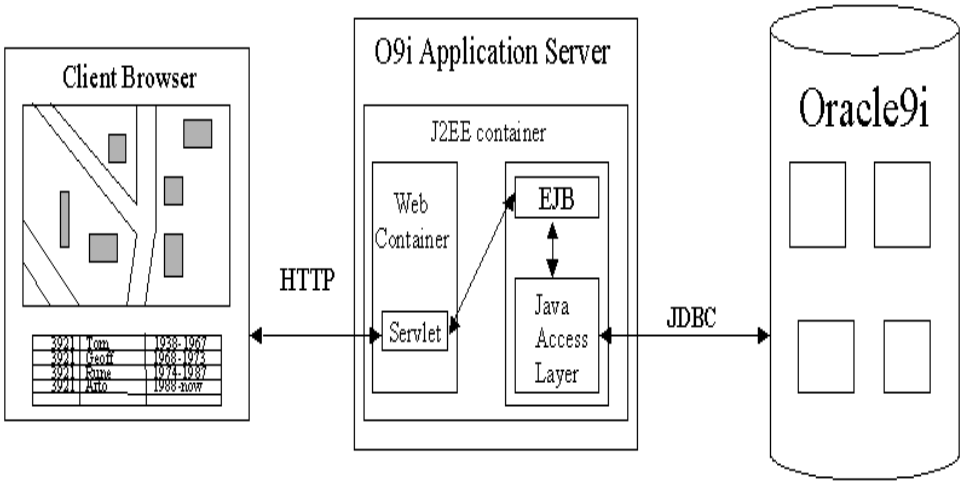


Figure 7.2: The demo architecture

Main application logic is found in EJB which uses a simple interface in the access layer to retrieve objects from the database. Communication with the database is provided with JDBC, which is a Java API for connectivity to database systems. The access layer contains a caching mechanism to avoid unnecessary transfer of objects from the database. Once the objects are loaded they reside in the application server, database access is only necessary to retrieve updated data. Based on the data retrieved, operations in the EJB generates a map of the spatial data and XML documents representing ownership history.

The servlet has a direct connection to the EJB and receives request from the client. That is, the servlet acts solely as a communication channel between the client and the EJB container.

The Map Client

The client accessing the application server is an applet. Applets are simple components running inside the environment of an Internet browser. The client applet was developed to access data from the application server and visualize it to the user. Main functionality of the client applet is:

- Display the generated map
- List ownership history for a given building
- Panning

The applet serves as a thin client and consists of a minimal amount of application logic. All complex information logic is found on the server-side. Using HTTP as communication to the application server a loose coupling between the application itself and the application logic is achieved. Loose coupling together with a thin client is favorable as almost any client easily can connect and utilize the application. This feature is strengthened since XML is used as interchange format. Loose couplings provide flexibility and portability for server and client connections since each tier is an autonomous unit.

7.6 Summary

The revised case model provides application developers with a clear and simple view of the modeled reality. The concept of valid time is introduced with well defined constructs, time related types and spatial extents are simply introduced by a set of data types. Constructs defined in the model are sufficient to generate difficult and complex code which in other cases must be done manually. In a software development purpose such generation will be of great help to the application developer. The developer can concentrate on designing the system model itself using fairly simple and defined concepts. Valuable time spent on modeling complex structures with ambiguous meaning is saved. Not to mention the time consumed by implementing the system model manually. The implemented case application has proved the above. Although the application architecture used is rather complex itself, the generated code provided us with a simple interface to complex functionality.

Chapter 8

Evaluation and Discussion

Our aim was to introduce temporal aspects as built in constructs to define a temporal profile for COMDEF. To simplify the process we narrowed the research area to an investigation of valid time as the temporal aspect. In this chapter we review COMDEF Temporal against the requirements from chapter 4 and compare with results of others. The comparison is discussed and we elaborate on the results.

8.1 Evaluation of Requirements

We compare COMDEF Temporal with three other temporal conceptual models. In section 2.2.4 we introduced TUML [75], TimeER [26] and the Extended Spatio-temporal UML, from now on ESTUML [56], as other temporal extensions to existing conceptual models. All the models represent different approaches. TUML is based on the Tau object model [36], while ESTUML has no such underlying model. TimeER is an extension of the EER model defined in [19] and has a defined mapping algorithm to the relational model [28]. Note, the following is not a complete evaluation of the three models, in the sense that the models are only used to point out differences and hence evaluate COMDEF.

8.1.1 The Time Model

In chapter 5 we discussed the diversity of time and defined a time model for COMDEF Temporal. Below is the COMDEF time model evaluated against the proposed requirements and compared with the other models. Table 8.1 shows the requirements we proposed for time models in chapter 4.

Linear (RT1)
All models support a linear model of time.

Time Model	COMDEF	TUML	ESTUML	TimeER
RT1	√	√	√	√
RT2	–	–	–	–
RT3	√	√	√	√
RT4	–	–	√	–
RT5	√	–	√	–
RT6	–	–	√	–

Table 8.1: Time model requirements revisited

Branching (RT2)
COMDEF does not support a branching model of time. A branching time line would imply significant changes to the model and support from underlying architecture. ESTUML allows a user defined time model, but does not explicitly state support for branching time. None of the last two models support branching time.

Discrete (RT3)
COMDEF adopts a discrete view of time. In general databases time has a discrete definition, a discrete view of time is therefore sufficient for most applications. All the other models define a discrete view of the timeline.

Continuous (RT4)
The diversity of ESTUML allow the modeler to chose between a continuous time line and a discrete time line. None of the other models support RT4.

Infinite (RT5)
In COMDEF the time line is infinite, and at the conceptual level, boundaries are set by the application. TUML and TimeER have both a bounded timeline by definition. In ESTUML, boundaries can be set explicitly in the model.

Circular (RT6)
Only ESTUML supports the notion of circular or periodic time, i.e. regular time using their notation.

Evaluation Summary
COMDEF defines a general model of time similar to temporal models found in literature, i.e. a linear, discrete and infinite time line. COMDEF, TUML and TimeER all have a fixed notion of time. ESTUML allows the definition of different time models using a specification box. Although COMDEF lacks a few concepts, the requirements fulfilled allows us to model the temporal

aspects of most discrete entities of real world domains.

8.1.2 The Temporal Model

We defined a temporal model in chapter 5 supporting the most common concepts related to valid time. Table 8.2 shows requirements for temporal models evaluated against COMDEF and compared with the three other models.

Temporal Model	COMDEF	TUML	ESTUML	TimeER
RTM1	√	√	√	√
RTM2	–	√	√	√
RDM3	√	√	√	√
RTM4	–	√	–	–
RTM5	√	√	–	√
RTM6	√	√	–	–
RTM7	–	–	–	√
RTM8	–	√	√	√
RTM9	√	√	√	–
RTM10	–	√	√	√
RTM11	√	√	√	√
RTM12	–	–	–	–
RTM13	√	√	–	–

Table 8.2: Temporal model requirements evaluated

Valid Time (RTM1)
All models support valid time.

Transaction Time (RTM2)
All models but COMDEF support transaction time.

User Defined Time (RTM3)
This feature is supported by all the surveyed models. COMDEF have defined the datetimes as described in the SQL standard to support user defined time.

Beginning (RTM4)
Only TUML has defined an explicit value denoting the first value on the time domain. COMDEF and ESTUML view this time value as in the infinite past without any further meaning to the model.

Now (RTM5)
COMDEF, TUML and TimeER support *now* as a current time variable.

Temporal variables are not included in ESTUML.

Forever (RTM6)
Only COMDEF and TUML include the variable *forever* as a concept. Both models support the notion of future valid time values.

Until Changed (RTM7)
TimeER supports *Until Changed* as a transaction time marker. COMDEF does not support transaction time.

Instant TimeStamp (RTM8)
TUML, TimeER and ESTUML use instant timestamps to denote event based semantics of attributes. TimeER defines interpolation functions for deriving values at points in time where no values are recorded when timestamping for valid time using instants. COMDEF does not define instant timestamps.

Interval TimeStamp (RTM9)
COMDEF, TUML and ESTUML support intervals as temporal primitives for timestamping facts. TimeER does not define interval timestamps for valid time.

Temporal Element TimeStamp (RTM10)
All except COMDEF support timestamping properties with temporal elements. Allowing COMDEF to timestamp with temporal elements implies significant changes to the model, also, the relational model does not support non-atomic attribute values.

Absolute Time (RTM11)
All the models associate facts with absolute time values. To be accurate, models recording absolute time values but involve now-relative data do in a sense support relative time. That is, facts are relative to the current time. Both COMDEF and TUML are defined within this group.

Relative Time (RTM12)
None of the models support relative time. Temporal reasoning mechanisms beyond those supported by conventional databases are necessary to support relative time. Knowledge based systems used by artificial intelligence applications are an example of systems supporting relative time [38].

Interval Operators (RTM13)
COMDEF has explicit operators defined in the model, each valid time entity have Allen's interval operators defined to compare the valid time timestamps. Similar operations for the timestamp types are defined in TUML.

Evaluation Summary
The evaluation above has proved the fact that designing a temporal model capturing all aspects is difficult. COMDEF use a simple approach, we define valid time and timestamps of type valid time intervals. Common temporal variables are defined and we support the interval comparison operators defined by Allen [1]. Although COMDEF does not support all the proposed requirements, most features common to temporal models are defined. The temporal model provides concepts necessary to define a temporal data model capturing the information history of entities.

8.1.3 The Temporal Data Model

Temporal concepts were introduced as an orthogonal extension to the existing framework in chapter 6. The result was a temporal data model supporting entity timestamping with intervals. Table 8.3 compares COMDEF Temporal with TUML, ESTUML and TimeER against the temporal data model requirements.

Data Model	COMDEF	TUML	ESTUML	TimeER
RDM1	-	√	√	√
RDM2	√	-	-	-
RDM3	-	√	√	√
RDM4	-	√	√	√
RDM5	√	√	√	-
RDM6	-	√	√	√

Table 8.3: Comparison of temporal data model requirements

Lifespan of Entities (RDM1)
COMDEF does not explicitly capture lifespan of an entity. Lifespan in COMDEF is implicit and deduced by the union of state timestamps for an entity. TUML, ESTUML and TimeER all have built in support for lifespan of modeled objects, i.e. the time an entity exists. ESTUML define existence time instead of lifespan, claiming a more exact definition as described in section 2.2.3.

Timestamped Entities (RDM2)
COMDEF timestamps entities as whole. We chose an entity timestamped approach due to the simplicity of representing such in a relational database. Although entity timestamping has drawbacks, such as the spreading of an entity into tuples and redundancy issues, the approach makes it simple to

model temporal entities for persistence in a model driven environment.

Timestamped Attributes (RDM3)
TUML, TimeER and ESTUML timestamp attributes. Both TUML and ESTUML are based on object models which have better support for an attribute timestamping scheme. TimeER though, is faced with the problem of representing multi-valued attributes in a database. Even if attribute timestamping represents a more natural approach when timestamping entities, it is difficult to implement using the relational model. Due to the restriction of atomic attribute values, a single entity must be split into several relations where each relation records e.g. the history of a single attribute. If a model supports multi-valued attributes, attribute timestamping allows the whole entity history to be stored in one single tuple. Such representational models are obtainable when using nested relations in the relational model, that is, where the *first normal form* (1NF) restriction is removed [19]. They can also be represented using complex objects in an object oriented database as defined in [2].

Timestamped Associations (RDM4)
COMDEF does not support timestamping of associations. All the other models support valid time for relationships between the objects. COMDEF leaves the treatment of relationships between temporal entities to the application.

Time-Invariant Identifiers (RDM5)
Each temporal entity in COMDEF has a primary key by definition. Constraints expressed in OCL are defined to ensure that the primary keys are time-invariant. Identifiers in ESTUML and TUML are general system provided object-identifiers, e.g. oid's which are independent of the time varying nature of a object. TUML has no notation for keys in the UML notation, although the Tau object model TUML is based on states that defined keys must be time-invariant. Keys in TimeER are time-varying, an entity is uniquely identified over time by a surrogate identifier.

Granularity (RDM6)
COMDEF does not support the specification of timestamp granularity in the models. The database determines the smallest granule available and the size to be used is determined by the application. In TimeER, TUML and ESTUML the modeler may select an appropriate granularity for the given temporal primitive.

Evaluation Summary
COMDEF differ from the other models with respect to timestamping, it is the only model timestamping objects, i.e. entities. The approach has its pros

and cons. The advantage is simple mapping to the conventional relational model. The main disadvantage is that an entity is spread into states, i.e. tuples in a database, which is a somewhat unnatural representation for an entity. However, in a model driven framework it is easy to provide uniform access to data using reasonable interfaces. The redundancy issue is harder to cope with, although a similar issue is present using an attribute timestamped scheme where keys are replicated for each time varying attribute.

The other surveyed models have a broader specter than COMDEF, and hence are more powerful. But power comes at a cost. We chose a simple model to support a range of applications which at the same time should be fairly easy to use. Providing a wide range of features, such as the ESTUML does, complicates the model. The temporal data model supports the requirements set by most applications and proves itself viable to model real world objects varying in valid time.

8.1.4 Model Driven Development

We have designed a temporal data model capable of modeling the time varying nature of information using UML. As discussed, a variety of temporal data models are present in literature, but none has ever been presented in a model driven environment. Our aim of research was to examine the modeling of temporal systems using a model driven approach. Below, the Temporal Profile is evaluated against the three other models with respect to requirements for temporal conceptual models in a model driven development setting.

MDD	COMDEF	TUML	ESTUML	TimeER
RMD1	√	√	√	√
RMD2	√	-	-	N.A.
RMD3	√	√	-	-
RMD4	√	-	√	√
RMD5	√	-	-	-
RMD6	√	√	-	√

Table 8.4: Comparison of Model Driven environment requirements

Methodology Support (RMD1)
COMDEF is compliant with the UML which is the industry standard, methodology support is therefore ensured. TUML and ESTUML have also recognized the benefits of a common modeling language and use UML. The TimeER is an extension to the EER model as defined in [19] and is thus a familiar notation to database modelers.

UML Profile (RMD2)
COMDEF has a defined metamodel which is an extension to the UML metamodel and thus is a UML Profile. COMDEF is superior to the other models in this respect. TimeER is not based on the UML, hence the requirement is not applicable (N.A). Neither TUML nor ESTUML explicitly define a UML profile. Both approaches use stereotypes to define the temporal data model, but especially ESTUML introduce constructs not in alignment with the UML. E.g. the artificial group symbol stereotype and the specification-box construct have no reference in the UML metamodel. Lifespan, timestamp type and granularity are expressed using tagged values in TUML, as depicted in the figure 2.2(b) in section 2.2.4. Tagged values are metadata regarding a model element only, and cannot contain values. The extension mechanism in UML is defined to refine reference metamodels and not add arbitrary modeling extensions. Such extensions lead to UML dialects hardly supported by tools and cannot be used in a model driven architecture.

Interpretable Mapping from Graphical Notation (RMD3)
COMDEF supports mapping and code generation from graphical notation. The graphical notation is mapped via XMI to CML and then used with the code generation tools in COMDEF. None of the other models supports this feature. Since COMDEF Temporal is a UML profile, the model conforms to the UML, and a mapping is therefore provided through the XMI format.

Platform Independent (RMD4)
COMDEF is not based on any particular platform or underlying architecture. The temporal profile is language neutral. The model is, however, designed for a mapping to the relational model, but this is not a requirement. TUML on the other hand is an extension to the ODMG object model and is therefore dependent on an ODMG compliant ODBMS [36]. ESTUML has no notion of an underlying platform and is a language neutral conceptual model. TimeER can be mapped to any implementation platform, a mapping algorithm to two non temporal relational models is described in [28].

Metamodel of Time and Temporal Concepts (RMD5)
COMDEF Temporal defines a metamodel of time and the temporal model. This adds additional power to the temporal profile, since all concepts are defined in the metamodel. Of the surveyed models COMDEF is the only model having this feature.

Simple and Expressive (RMD6)
As it is difficult to evaluate the simplicity and expressiveness of a model objectively, we consider this requirement to be vague and not well-defined.

However, we try to express our informal view. COMDEF introduces a simple temporal data model, which features are limited in respect to other models, but the model is expressive enough. Modeling temporal entities for persistence is easy using the profile. Modeling is clean, the concepts are fairly simple to understand and the temporal support is well defined. In the case of ESTUML, the modeler needs comprehensive insight into temporal concepts and the notation to design meaningful models. Although the ESTUML is expressive and supports a wide specter of features, we regard the notation to be too complicated. Past experience has shown that models having high levels of expressive power lead to rejection by users due to complexity [54].

Evaluation Summary
The comparison of the models has proved that COMDEF supports model driven development. COMDEF defines a temporal UML profile conforming to the UML metamodel. Temporal semantics are introduced as well defined concepts in the COMDEF metamodel. To the degree possible, all temporal concepts are present in the metamodel and OCL expressions are used to define restrictions on the model elements. Compared to the other models, COMDEF is the only model having a complete metamodel for time and temporal concepts. Mappings are provided through the XMI format capturing all constructs in the model. In this way models designed by COMDEF Temporal is interpretable and can be used with code generation tools.

A common problem regarding temporal conceptual models is the non standardized extensions used to define the temporal data model. Even if the UML is a flexible language, we have seen that especially ESTUML have constructed artificial notations not in alignment with the UML to express temporal properties of model elements. In a model driven architecture such extensions out of alignment with the UML are not applicable. COMDEF Temporal is a UML profile and thus conforms to the standard, and from a Model Driven Development perspective with automatic code generation COMDEF is superior to all the others.

8.2 Summary

This chapter has evaluated the Temporal profile against the requirements from chapter 4. A comparison between the COMDEF and three other temporal data models have been performed. We have recognized COMDEF Temporal as a simple, but at the same time sufficiently expressive temporal data model, having the necessary concepts to model temporal persistent entities. COMDEF Temporal lacks temporal features represented in the other models, but the model has a great advantage in the support for model driven development. As we saw above, COMDEF fulfilled all the requirements for model driven development. This reflects the fact that COMDEF Temporal

was designed to support the Model Driven Architecture. As far as we are aware of, no other temporal data model supports the platform independent MDA.

Chapter 9

Conclusion and Further Work

In the following an overall summary and contributions of the thesis are presented before we conclude with areas for further work.

9.1 Concluding Remarks

At present, alternative ways of supporting temporal applications are the practice, tools supporting the modeling and implementation of temporal systems are valuable, but are lacking in the industry. It is within this domain this thesis contributes.

Our task has been to support a model based development process for temporal information systems by introducing temporal concepts with well defined temporal semantics to the models. In this way models with temporal concepts and semantics capable of representing a view as complete as possible of the modeled world can be designed. Such models can be utilized by tools and hence the semantics defined can by code generation substantially help to simplify the implementation process of temporal information systems. In a domain like temporal data management where the systems are complex and difficult to manage, this is of great benefit to application developers. Development time is reduced and system quality improved since systems are developed according to the intended design.

9.2 Contributions

This thesis has presented a temporal extension to a model driven development framework. In the introduction we described guidelines and goals we have tried to achieve for a temporal data model in a model driven environment. The main goal has been to design a temporal modeling language

having concepts with sufficient semantics to provide tools for code generation of systems capturing the semantics of temporal data. Contributions meeting or partly meeting the goal are listed below.

The Temporal profile - COMDEF Temporal is a temporal UML profile.
Metamodel capturing all concepts - All concepts are defined in the metamodel.
Precise definition of model elements - Precise metamodeling and OCL are used to define the profile.
Platform Independent Tool Support - Models designed with COMDEF Temporal are platform independent and code generation to a variety of platforms is possible.
Simple and User Friendly - The profile is easy to comprehend and simple to use.
Modeling for Persistence - Using COMDEF Temporal it is easy to model persistent database entities.

The Temporal Profile

We have introduced orthogonal valid time constructs to an existing UML profile and designed a temporal data model supporting entity timestamping for valid time with intervals. The result is a temporal UML Profile capturing the time varying nature of entities in a model. COMDEF Temporal is a well designed profile which conforms to the UML metamodel. The extension mechanisms used are stereotypes and constraints to specify the new model elements. Mapping rules to the lexical language CML are defined which allow code generation of models. A well defined profile for a specific domain has many advantages in a model driven environment, in the following we point out more specific contributions.

Metamodel capturing all concepts

By leaving as much semantics as possible in the model, we have proved some of the ideology behind the MDA. We have managed to express the temporal concepts of entities using UML. The model of time and the temporal model defined in the metamodel give a clear and intuitive view of the temporal aspect of the model. For a modeler it is important to know exactly how a modeling element is defined. For temporal conceptual models this is often omitted and usually described using natural language. Our approach shows a clearer definition than other temporal modeling languages. The semantics of the timestamp is described using modeling notation and constraints. This feature strengthens the profile. In a model driven architecture small nuances and supported features in the metamodel have large impacts on the generated code.

Precise definition of model elements
Precise definitions of modeling elements are vital in a model driven architecture. Unfortunately, the precision of UML is not very strong, the language is not precisely defined itself. We have striven to define the temporal profile as exactly as possible, using precise metamodeling and OCL to constrain the model elements. We consider the goal only partly fulfilled, both due to the lack of precision of UML, and the further investigation of temporal constraints using OCL. The UML itself is not explicit enough to define precise models, extensive use of OCL and in some cases more formal notations like first order logic are necessary to express the semantics.

Platform Independent Tool Support
When developing systems, abstraction should be lead away from platforms and language issues. Solving the problem should be of main concern. Modeling frameworks should therefore be language, platform and product independent. A problem with specialized domains such as temporal and e.g. spatial data is that they are usually managed by dedicated and monolithic systems. CASE tools used for design and implementation of such systems are bound to specific design tools and architectures. Or, as we saw in our evaluation, models use non-standard extensions to standardized languages creating dialects not interchangeable.

The COMDEF Temporal solves this problem by conforming to the Model Driven Architecture. UML compliance is assured which means that the profile can be used in any modeling tool supporting the UML. The profile is platform independent and hence models can be mapped to any architecture. For a domain like temporal data management this is a great achievement. Temporal database application developers can benefit substantially from the Temporal Profile in a mature MDA environment.

Simple and User Friendly
The problem with temporal conceptual models, in the same way as with temporal models, is the amount of criteria defined. Models are loaded with features making it hard for users to comprehend and learn the language. Models fail to visualize a correct view of the modeled world. We have defined a simple model with few requirements which are well defined, our view is that expressive simplicity is necessary to develop a temporal data model. The resulting temporal extension captures more conveniently temporal aspects of information systems than the regular UML does. Modeling is expressive and clean, no properties are linked to objects by artificial constructs which tend to obscure diagrams. The model concepts are simple, but still retain enough expressive power and semantics. Using COMDEF temporal it is simple to model persistent entities with valid time support.

Modeling for Persistence
COMDEF Temporal allows the modeling of persistent entities using UML. Many companies and organizations have sought for a UML profile for persistent relational database modeling. The COMDEF DB contributes with persistent modeling capabilities for the UML.

9.3 Further Work

This section presents topics for further work. A variety of areas are discovered that should receive closer attention. Topics refer both to temporal modeling and model driven development issues.

Continue work on temporal data model

Due to the time constraints of a master thesis a variety of topics are not fully explored. Structural properties of the temporal data model have been the main topic, that is, the design of the temporal metamodel. Operations on the data model have not been defined and would contribute to a more complete temporal data model. E.g. time slice operations returning entity states based on an instant or interval.

An unambiguous UML

We have discussed shortcomings of the UML, in special the extension mechanism. A variety of other problems have also been recognized, among them the lack of formal definition of the language. The Perfect UML (PUML) group is a collection of researchers working on the topic of rearchitecturing UML for the future. In [11] they define a new UML infrastructure based on formal definitions of all modeling constructs. In short they introduce patterns encoded as templates to handle the definition of languages. The approach, based on the encoding of template packages and template package generalization, allows package merging and renaming. Constructing new metamodels will be easier and less complicated using this method. Instantiation of templates allows factoring of language components with exact specification, complete metamodels can be constructed quickly with a precise, unambiguous meaning. Research concerning the use of this method for defining the Temporal Profile has been done, and the method proves as a flexible and precise way to define metamodels. This topic should be further examined, as mentioned above, a precise UML results in more semantically correct metamodels which again lead to a greater degree of code generation.

Consistency of a model

Developers make mistakes. It is therefore important that the believed miniworld modeled actually conforms to the semantics of the constructs used

to build it. Consistency of a model regarding a given domain specific metamodel is important in a model driven context. Having a tool that checks the consistency of a model against the metamodel along with the correctness of OCL expressions would be a great achievement. Code generation tools from the parsing of OCL expressions should also be a common part of a UML modeling tool. In this way constraints defined using OCL can be evaluated at runtime by the generated code. COMDEF can be extended to support such tools by including a model checker and an OCL compiler as a front-end to the code generation tools. OCL compilers have been designed, one example is the Dresden OCL Toolkit [17].

Transaction Time Support

Transaction time is not defined for COMDEF Temporal. Some work on transaction time support for entities have been examined, e.g. the generation of audit-trails for entities is an example mapping which is available. This work should be continued and further investigated to add transaction time support for the temporal profile. Combining the two dimensions to define bitemporal support for entities is another interesting topic.

Associations between temporal entities

The semantics of associations in a temporal modeling language needs careful attention. Two options to consider are whether the associations should be timestamped explicitly or if the time an association is valid in the modeled reality is deduced from the lifespan of the participating objects. An extension to the COMDEF relationship model to support timestamping of associations requires a considerable amount of work.

Bibliography

[1] James F. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23:123–154, 1984.

[2] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In *In Proceedings of the First International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan*, pages 223–40, December 1989.

[3] Stefan Berner, Martin Glinz, and Stefan Joos. A Classification of Stereotypes for Object-Oriented Modeling Languages. In Robert France and Bernhard Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723, pages 249–264. Springer, 1999.

[4] Elisa Bertino, Elena Ferrari, and Giovanna Guerrini. A Formal Temporal Object-Oriented Data Model. In *Proceedings of the 5th International Conference on Extending Database Technology (EDBT)*, Avignon, France, 1996.

[5] M.H. Böhlen, R. Busatto, and C.S. Jensen. Point- Versus Interval-based Temporal Data Models. In *In Proceedings of the 14th International Conference on Data Engineering*, pages 192–200, 1998.

[6] Michael H. Böhlen and Christian S. Jensen. Seamless Integration of Time into SQL. Technical Report R-96-49, Department of Computer Science, Aalborg University, 1996.

[7] Michael H. Böhlen, Richard T. Snodgrass, and Michael D. Soo. Coalescing in Temporal Databases. In *Proceedings of the 22nd International Conference on Very Large Databases*, pages 180–191, Bombay, India, September 1996.

[8] Grady Booch, James Rumbaugh, and Ivar Jacobsen. *The Unified Modeling Language User Guide.* Addison-Wesley Longman, Inc., 1999.

[9] R. G. G. Cattell et al., editors. *Object DataBase Standard: ODMG 2.0.* Morgan KaufMann Publishers, Inc, San Francisco, 1997.

[10] Jan Chomicki. Temporal Query Languages: a survey. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic: ICTL'94*, volume 827, pages 506–534. Springer-Verlag, 1994.

[11] Tony Clark, Andy Evans, Robert France, Stuart Kent, and Bernard Rumpe. pUML response to UML2.0 RFI, 1999. Available at http://www.puml.org.

[12] Tony Clark, Andy Evans, Stuart Kent, Steve Brodsky, and Steve Cook. A Feasibility Study in ReArchitecting UML as a Family of Languages using a Precise OO Meta-Modeling Approach. Available from http://www.puml.org, September 2000.

[13] J. Clifford, A. Croker, and A. Tuzhilin. On Completeness of Historical Relational Query Languages. *ACM Transactions on Database Systems*, 19(1):64–116, March 1994.

[14] James Clifford, Curtis Dyreson, Tomas Isakowitz, Christian S. Jensen, and Richard T. Snodgrass. On the Semantics of "Now" in Databases. *ACM Transactions on Database Systems*, 22(2):171–214, June 1997.

[15] COMBINE. Project website. http://www.opengroup.org/combine, 1/2/02.

[16] Linda G. DeMichiel, L. Ümit Yalçinalp, and Sanjeev Krishnan. Sun Microsystems: Enterprise Javabeans Specification Version 2.0, October 2000. Available at http://java.sun.com/products/ejb.

[17] The Dresden OCL Toolkit. Project website. http://dresden-ocl.sourceforge.net/,13/3/02.

[18] Curtis E. Dyreson, Michael D. Soo, and Richard T. Snodgrass. The Data Model for Time. In Snodgrass [65], chapter 6.

[19] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems.* The Benjamin/Cummings Publishing Company, Inc, 1994.

[20] Ramez Elmasri, Gene T. J. Wuu, and Vram Kouramajian. A Temporal Model and Query Language for EER Databases. In Tansel et al. [74], chapter 9.

[21] Martin Erwig, Ralf H. Güting, Markus Schneider, and Michalis Vazirgiannis. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *GeoInformatica*, 3(3):269–296, 1999.

[22] S. K. Gadia. A Homogenous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13(4):418–488, December 1988.

[23] Shashi K. Gadia and Sunil S. Nair. Temporal Databases: A Prelude to Parametric Data. In Tansel et al. [74], chapter 2.

[24] Robert Geisler. Precise UML Semantics Through Formal Metamodeling. In Luis Andrade, Ana Moreira, Akash Deshpande, and Stuart Kent, editors, *Proceedings of the OOPSLA'98 Workshop on Formalizing UML. Why? How?*, 1998.

[25] Iqbal A. Goralwalla, M. Tamer Ozsu, and Duane Szafron. A Object Oriented Framework for Temporal Data Models. In Opher Etzion, Sushil Jajodia, and Suryanarayana Sripada, editors, *Temporal Databases: Research and Practice*, volume 1399, pages 1–35. Springer-Verlag, 1998.

[26] Heidi Gregersen and Christan Jensen. Conceptual Modeling of Time-Varying Information. Technical Report TR-35, TimeCenter, September 1998.

[27] Heidi Gregersen and Christian S. Jensen. Temporal Entity-Relationship Models-a Survey. Technical Report TR-3, TimeCenter, 1997.

[28] Heidi Gregersen, Leo Mark, and Christian S. Jensen. Mapping Temporal ER Diagrams to Relational Schemas. Technical Report TR-39, TimeCenter, December 1998.

[29] L.H. Hermosilla. A Unified Approach for Developing a Temporal GIS with Database and Reasoning Capabilities. EGIS(1994), http://www.odyssey.maine.edu/gisweb/spatdb/egis/eg94014.html.

[30] Christian S. Jensen and Curtis Dyreson (eds). The Consensus Glossary of Temporal Database Concepts -February 1998 Version. Available at http://www.cs.auc.dk/~csj/Glossary/, 1998.

[31] Christian S. Jensen and Richard T. Snodgrass. Semantics of Time-Varying Information. *Information Systems*, 21(4):311–352, 1996.

[32] Christian S. Jensen and Richard T. Snodgrass. Temporal Data Management. Technical Report TR-17, TimeCenter, June 1997.

[33] Christian S. Jensen, Michael D. Soo, and Richard T. Snodgrass. Unifying Temporal Data Models Via a Conceptual Model. *Information Systems*, 19(7):513–547, 1994.

[34] Wolfgang Käfer, Norbert Ritter, and Harald Schöning. Support for Temporal Data by Complex Objects. In Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek, editors, *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*, pages 24–35. Morgan Kaufmann, 1990.

[35] Ioannis Kakoudakis and Babis Theodoulis. The TAU Time Model. Technical Report TR-96-5, TimeLab, October 1996.

[36] Ionnais Kakoudakis and Babis Theodoulidis. The TAU Temporal Object Model. Technical Report TR-96-5, TimeLab, October 1996.

[37] Won Kim. Introduction to Part1: Next-Generation Database Technologuy. In *Modern Database Systems, The Object Model, interoperability and beyond*. Addison-Wesley Publishing Company,Inc., 1995.

[38] B. Knight and M. Jixin. Temporal management using relative time in knowledge-based process control. *Engineering Applications of Artificial Intelligence*, 10(3):269–280, June 1997.

[39] Chris Kobryn. UML2001: A Standardization Odyssey. *Communications of the ACM*, 42(10), October 1999.

[40] Curtis P. Kolovson. Indexing Techniques for Historical Databases. In Tansel et al. [74], chapter 17.

[41] Gunnar Lunde. Automated GUI-Development. Master's thesis, The University of Oslo,Institute of Informatics, 2002.

[42] Shridhad Lyengar. Implementing Metamodels and Repositories using the MOF. In Jon Siegel, editor, *CORBA 3 Fundamentals and Programming*, chapter 22. Wiley Computer Publishing, John Wiley & and Sons, Inc, 2000.

[43] Jim Melton and Alan R. Simon. *Understanding the new SQL: A Complete Guide*. Morgan Kaufmann Publishers, Inc., 1993.

[44] Joaquin Miller, Jishnu Mukerji, et al. Model Driven Architecture (MDA). OMG, Document number ormsc/2001-07-01, July 2001. Available at http://www.omg.org/mda.

[45] Angelo Montanari and Barbara Pernici. Temporal Reasoning. In Tansel et al. [74], chapter 21.

[46] Bjørn Nordmoen. Beyond CORBA Model Driven Development. Schlumberger Oslo Technology Center, available at http://www.omg.org/mda, 15/02/02, 2001.

[47] OBOE. The OBOE project website. http://www.opengroup.org/oboe, 1/2/02.

[48] OMG. The Common Object Request Broker: Architecture and Specification. Object Managment Group, Revison 2.0, July 1996. Available at http://www.omg.org.

[49] OMG. Meta Object Facility (MOF) Specification, Version 1.3, March 2000. Available at http://www.omg.org.

[50] OMG. Unified Modeling Language Specification, Version 1.3, March 2000. Available at http://www.omg.org.

[51] OMG. XML Metadata Interchange (XMI) specification, Version 1.0, June 2000. Available at http://www.omg.org.

[52] Oracle. Oracle8i Spatial User's Guide and Reference, February 1999. Available at www.oracle.com.

[53] Gultekin Ozsoyoglu and Richard T. Snodgrass. Temporal and Real-Time Databases: A Survey. *IEEE Transactions On Knowledge And Data Engineering*, 7(4):513–532, August 1995.

[54] Christine Parent, Stefano Spaccapietra, and Esteban Zimányi. Spatio-temporal conceptual models: Data structures + space + time. In Claudia Bauzer Medeiros, editor, *ACM-GIS '99, Proceedings of the 7th International Symposium on Advances in Geographic Information Systems, November 2-6, 1999, Kansas City, USA*, pages 26–33. ACM, 1999.

[55] Niki Pissinou et al. Toward an Infrastructure for Temporal Databases: Report of an Invitational ARPA/NSF Workshop. *SIGMOD*, 23(1):35–51, 1994.

[56] Rosanne Price, Nectaria Tryfona, and Christian Jensen. A Conceptual Modeling Language for Spatiotemporal Applications. Technical Report CH-99-20, Chorochronos, 2000.

[57] The TOOBIS project. TODM, Specification and Design. Available at http://www.mm.di.uoa.gr/~toobis/, dec 1996.

[58] M-J Proulx and V. Bedard. Perceptory website. (10/01/02), Perceptory, Centre de recherche en geomatique, Universite Laval, Quebec, Canada. http://sirs.scg.ulaval.ca/Perceptory.

[59] Markus Schneider. *Spatial Data Types for Database Systems*, volume 1288 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[60] Fabio A. Schreiber. Is Time a Real Time? - An Overview of Time Ontology in Informatics. In Wolfgang A. Halang and Alexander D. Stoyenko, editors, *Real Time Computing*, pages 283–307. Springer Verlag NATO-ASI Vol. F127, 1994.

[61] John Siegel et al. Developing in OMG's Model-Driven Architecture. OMG White Paper, Revision 3.2, November 2001. Available at http://www.omg.org/mda.

[62] Bjørn Skjellaug. Temporal Data: Time and Relational Databases. Research Report 246, Department of Informatics, University of Oslo, April 1997.

[63] Richard Snodgrass. The temporal query language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[64] Richard T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann Publishers, 2000.

[65] Richard Thomas Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 1995.

[66] Arnor Solberg, Tor Neple, Jon Oldevik, and Bård Kvalheim. A Flexible Framework for Development of Component-based Distributed Systems. Sintef Telecom and Informatics, Distributed Information Systems, 1999.

[67] Richard Soley et al. Model Driven Architecture. OMG White Paper, Draft 3.2, November 2000. Available at http://www.omg.org/mda.

[68] Stefano Spaccapietra, Christine Parent, and Esteban Zimany. Modeling Time from a Conceptual Perspective. In J. French and G. Gardarin, editors, *Proc. of the 7th Int. Conf. on Information and Knowledge Management, CIKM'98*, pages 432–440. ACM Press, November 1998.

[69] A. Steiner and M. C. Norrie. A Temporal Extension to a Generic Object Data Model. Technical Report 265, Institute for information Systems,ETH Zürich,Switzerland, May 1997.

[70] Andreas Steiner and Moira C. Norrie. Implementing Temporal Databases in Object-Oriented Systems. Tr-14, TimeCenter, may 1997.

[71] Andreas Steiner and Moira C. Norrie. Temporal Object Role Modelling. In Antoni Olivê and Joan Antoni Pastor, editors, *Advanced information Systems engineering 9th International Conference, CAiSE'97 Barcelona, Catalonia, Spain, June 1997 Proceedings*, volume 1250, pages 245–258. Springer-Verlag, 1997.

[72] Stanley Y. W. Su and Hsin-Hsing M. Chen. A Temporal Knowledge Representation Model OSAM*/T and Its Query Language OQL/T. In Guy M. Lohman, Amílcar Sernadas, and Rafael Camps, editors, *17th International Conference on Very Large Data Bases, September 3-6, 1991, Barcelona, Catalonia, Spain, Proceedings*, pages 431–442. Morgan Kaufmann, 1991.

[73] Sun. Java^{TM}2 Platform, Enterprise Edition Specification Version 1.3. Copyright 1999-2000, Sun Microsystems, Inc. Available at http://java.sun.com/j2ee/docs.html.

[74] A.U. Tansel, J.Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal Databases: Theory, Design and Implementation*. The Benjamin/Cummings Publishing Company Inc., 1993.

[75] Babis Theodoulidis and Marianthi Svinterikou. Temporal Unified Modelling Language(TUML). Technical Report CH-97-11, TimeLab, November 1997.

[76] Charalampos I. Theodoulidis and Pericles Loucopoulos. The Time Dimension in Conceptual Modelling. *Information Systems*, 16(3):273–300, 1991.

[77] Babis Theoudoulidis. Towards the First Generation of Temporal Information Servers. In R. T. Snodgrass, editor, *In Proceedings of the 1st International Workshop on an Infrastructure for Temporal Databases, Arlington, Texas*, June 1993.

[78] Kristian Torp, Christian S. Jensen, and Richard T. Snodgrass. Modification Semantics in Now-Relative Databases. Technical Report TR-43, TimeCenter, September 1999.

[79] W3C. Extensible Markup Language (XML) 1.0, February 1998. Available at http://www.w3c.org.

[80] W3C. XSL Transformations (XSLT) Version 1.0, November 1999. Available at http://www.w3c.org.

[81] Tomas Wangen. Data Intensive Types. Master's thesis, The University of Oslo,Institute of Informatics, 2002. Draft.

[82] Jos Warmer and Anneke Kleppe. *The Object Constraint Language Precise Modeling with UML*. Addison Wesley Longmann, Inc., 1999.

[83] Michael F. Worboys. *GIS: A Computing Perspective*. Taylor & Francis, 1995.

[84] Gene T. J. Wuu and Umeshwar Dayal. A Uniform Model for Temporal and Versioned Object-oriented Databases. In Tansel et al. [74], chapter 10.

[85] Jun Yang, Huacheng C. Ying, and Jennifer Widom. Tip: A temporal extension to informix. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, volume 29, page 596. ACM, 2000.