A TEMPORAL OBJECT RELATIONAL DATABASE SYSTEM FOR 3D OBJECTS

VO THI NGOC CHAU

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
DOCTOR OF ENGINEERING IN ELECTRICAL ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2008

KMITL-2008-EN-D-018-393

หัวข้อวิทยานิพนธ์	ระบบฐานข้อมูลเชิงเวลาชนิดวัตถุสัมพันธ์สำหรับวัตถุ3มิติ
นักศึกษา	Vo Thi Ngoc Chau
รหัสนักศึกษา	48060021
ปริญญา	วิศวกรรมศาสตรดุษฎีบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2551
อาจารย์ ที่ปรึกษาวิทยานิพนธ์	รศ. ดร. ศุภมิตร จิตตะยโศธร

บทคัดย่อ

ในปัจจุบันนี้ ข้อมูลที่เป็นเรขภาพ3มิติได้ถูกใช้งานอย่างกว้างขวาง มีเครื่องมือซอฟท์แวร์สนับสนุนเพื่อสร้างและปรับปรุงแก้ไขเรขภาพ3มิติ อย่างไรก็ตามการจัดการข้อมูล3มิติจำนวนมากเหล่านี้ให้สนับสนุนการเปลี่ยนแปลงเชิงเวลา สามารถเข้าถึงและใช้งานได้โดยผู้ใช้จำนวนมาก และมีข้อมูลเมต้าเชิงเวลาเป็นเรื่องที่น่าสนใจ

วิทยานิพนธ์นี้นำเสนอระบบฐานข้อมูลเชิงเวลาชนิดวัตถุสัมพันธ์สำหรับวัตถุ3มิติ เพื่อเป็นการสนับสนุนการจัดการข้อมูล3มิติดังกล่าว เราได้นำเสนอรูปแบบการจัดการฐานข้อมูลเชิงเวลาในระดับแอททริบิวท์ และนำเสนอภาษาเอสคิวแอลเชิงเวลาที่สามารถจัดการข้อมูลเชิงเวลาในระดับแอททริบิวท์ ภาษาดังกล่าวประกอบไปด้วยส่วนภาษานิยามข้อมูล เพื่อสร้างและเปลี่ยนแปลงโครงสร้างข้อมูล กฎสำหรับบังคับบูรณภาพของข้อมูล ภาษาค้นคืนข้อมูล และคำสั่งสำหรับปรับปรุงแก้ไขข้อมูล ภาษาเอสคิวแอลเชิงเวลาดังกล่าวได้รับการออกแบบและพัฒนาบนระบบจัดการฐานข้อมูลเชิงวัตถุสัมพันธ์

นอกจากนี้วิทยานิพนธ์นี้ยังนำเสนอแบบจำลองข้อมูลเชิงแนวคิดสำหรับออกแบบฐานข้อมูลเชิงเวลา โดยปรับปรุงแบบจำลองข้อมูลไนแอมให้รองรับข้อมูลเชิงเวลาด้วย แบบจำลองข้อมูลดังกล่าวถูกนำไปใช้ในการนำเสนอข้อมูล3มิติเชิงเวลาได้อย่างเหมาะสมและถูกใช้เป็นข้อมูลเมต้าสำหรับเรขภาพ3มิติ แบบจำลองเชิงแนวคิดดังกล่าวสามารถได้รับการแปลงโครงสร้างข้อมูลเชิงแนวคิดเป็นโครงสร้างข้อมูลระดับตรรกเพื่อใช้งานกับข้อมูลทั่วไปและข้อมูลเรขภาพ3มิติบนระบบฐานข้อมูลเชิงเวลาชนิดวัตถุสัมพันธ์ที่ได้พัฒนาขึ้นได้

Thesis Title	A Temporal Object Relational Database System for 3D Objects
Student	Miss. Vo Thi Ngoc Chau
Student ID.	48060021
Degree	Doctor of Engineering
Program	Electrical Engineering
Year	2008
Thesis Advisor	Assoc. Prof. Dr. Suphamit Chittayasothorn

ABSTRACT

Along with audio/visual data, three dimensional (3D) graphical data has been popular in a wide range of application domains due to good effects of 3D visualization and the available support from recent advances in computer science by means of 3D data acquisition and authoring tools. A large amount of 3D graphical data needs to be effectively managed for convenient access, sharability, and reusability. Moreover, once 3D graphical data is time-varying, the temporal aspect of the 3D graphical data and metadata surrounding them should be captured to reflect changes along time.

In order to deal with 3D objects over the time, we develop a temporal object relational database system for 3D objects whose properties, relationships, and/or themselves are supported with valid time. The object relational database model with the valid time support at both object and attribute levels is defined to represent both graphical data and metadata parts of 3D objects. Accompanying this temporal object relational database model, a temporal object relational SQL language is innovatively proposed as a temporal extension of the standard object relational SQL language in association with temporal logic for ease and intuition. The language allows the nonprocedural temporal specification of data definitions (schema definitions, schema evolutions, and integrity constraints), querying, and modifications (insertions, deletions, and updates). It serves a large family of both non-temporal and temporal database users at several various complexity levels of temporal data manipulations in a temporal transparency environment. Based on this temporal object relational database model, a temporal object relational database is determined for both graphical data and metadata

parts of 3D objects. In addition, the temporal object oriented NIAM (Nijssen's Information Analysis Methodology) conceptual schema model is proposed to represent metadata about 3D objects in respect of valid time at the abstract level. A temporal object oriented NIAM conceptual meta schema is formed after our automatic data extraction process on an input 3D graphical data source. A conceptual/logical transformation procedure is systematically invoked to obtain a temporal object relational meta database schema. Then, both graphical data and metadata of 3D objects can be captured with regard to valid time by means of our temporal data population process.

As soon as a temporal object relational database for 3D objects is available, front-end applications can have access to 3D objects exactly and conveniently by querying the graphical data and metadata parts of these 3D objects in many aspects: graphics/semantics, non-temporal/temporal, non-spatial/spatial, and non-presentable/presentable with the use of the proposed temporal object relational SQL language. In comparison with related works, only our system examines the temporal aspect of metadata about 3D objects. As a result, the graphical data and metadata about 3D objects over the time can be consumed in a standard way. 3D objects can be easily reached along time at both graphical and semantic data levels for sharability and reusability.

Acknowledgements

First of all, I would like to thank all people who have supported and helped me to obtain the scholarship under the AUN/SEED-net project so that I was offered a good opportunity to be here for my higher education and to approach interesting cultures, new knowledge, and good friends from other countries.

Without a great deal of help and a great number of advices from my academic advisor, Assoc. Prof. Dr. Suphamit Chittayasothorn, this research wouldn't have been completed. I highly appreciate not only his time but also his support. He is very patient to listen to what I attempt to present during my research period. Furthermost, he is so kind that he always forgives me and gives me other good chances if I make mistakes or lose my ways. From my deep respects, I would like to express my special thanks to him.

Also, all knowledge I achieved at this degree comes from many lecturers at Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand. At this moment, I might not take advantage of such courseworks as much as the lecturers expected. However, all courseworks helped me train my thinking and they will be necessary for my future work hopefully. For their willingness and kindness, I want to thank all the lecturers very much.

Further, many special thanks go to Prof. Dr. Shigeyuki Ohara for his advices on the 3D applications of this research work under the Southeast Asia Engineering Education Development Network (SEED-Net) Project of the ASEAN University Network (AUN). I also greatly appreciate his kind consideration and time for my visit and study in his lab at the Tokai University. In addition, I would like to thank Assoc. Prof. Dr. Kazuhiko Hamamoto very much for the introduction on his research works in the virtual reality arena.

And I would like to thank very much my former supervisor and other lecturers at Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology, Vietnam, together with all of my friends including Vietnamese friends and non-Vietnamese friends, for their kindness and encouragement, which they gave me directly as well as indirectly.

Finally, this research work is all dedicated to my beloved Parents, Sister, and Brothers.

Contents

Contents (cont.)

Contents (cont.)

Contents (cont.)

Contents (cont.)

List of Tables

List of Tables (cont.)

Table Page

List of Figures

List of Figures (cont.)

List of Figures (cont.)

List of Figures (cont.)

List of Figures (cont.)

Chapter 1

Introduction

With recent advances in computer science and engineering, audio/visual data as well as three dimensional (3D) graphical data has been easily generated and spread over a wide range of application domains such as architecture, real estate, interior design, education, and medicine, etc. 3D objects can be acquired from real objects or from 3D data modeling with 3D programming toolkits, 3D authoring tools, and 3D data description languages. With such available supports, a large number of 3D objects exist and need to be effectively managed for convenient access, sharability, and reusability.

Moreover, the significance of using metadata is well accepted for the machine understanding of multimedia data, geographical information, and other resources. This fact is supported by the investigation of some international standards such as MPEG-7 [1], ISO 19115:2003 [2], and the Dublin Core metadata standard [3], respectively. Indeed, [4] introduces the use of semantic information in CAD applications for the improvement of the workflow and capabilities of the applications mainly dedicated to non-experienced users. Several advantages of using semantic information based on ontologies are also listed in [4] for management, searching, and sharing, etc. With more consideration for 3D graphical objects, [5, 6, 7, 8, 9, 10, 11, 12, 13] realize the need of metadata surrounding 3D graphical objects. This awareness stems from the lacking of support for semantic information in 3D data formats that are only invented with graphics terms. For instance, what 3D objects of interest in the Universe of Discourse (UoD) are in a scene, what semantic types of those objects are, what relationships, e.g. composition and spatial relationships (left, above, disjoint, etc.), among those objects are, and so on are not explicitly available along with the graphical data part of 3D objects and scenes.

Besides, temporal aspects are the inherent part of any entity. It is found that once 3D objects and the scenes in which they participate vary along time, metadata about them is time-varying as well. Thus, it is valuable to take into account the temporal aspect of 3D objects and metadata surrounding them to reflect their time-varying nature.

As for temporal database management, a large number of proposals have been investigated based on a few existing non-temporal data models: relational [14], nested relational [15], object oriented [16], and object-relational [17, 18]. Two main temporal aspects of data, valid time and transaction time [19], are determined. Temporal data representation has been formed with two different timestamping schemes: tuple/object and attribute. The former scheme regards the timestamp as a special attribute(s) of a relation scheme or an object class/type and hence as part of every tuple/object while as part of attributes individually and not of the whole tuple/object for the latter [20]. Temporal data manipulation has been supported by many temporal extensions to relational algebra/tuple calculus, QUEL, OQL, and SQL. Also related to time, temporal logic [21] which is a classical logic augmented with temporal operators to describe how truth values of assertions change over time has been researched for a long time. Due to its expressive power, it is useful in natural language processing, artificial intelligence, and computer science. Hence, we take advantage of temporal logic to set up the mathematical foundation of our temporal data querying extension and to allow users to non-procedurally express temporal queries against a temporal database by means of temporal operators (ALWAYS, ANYTIME, NEXT, PREV, and UNTIL).

In short, this research will bring database technologies to the handling of 3D objects along time.

1.1 Motivation

In the context introduced above, this subsection addresses the motivations of our work for the handling of 3D objects with respect to valid time.

- 3D graphical data is of our interest because of its existence in a wide range of application domains.
- Existing works that pay only attention to 3D objects normally define content-based retrieval techniques to make a search for individual 3D objects handled in an arbitrary manner. Their outcomes are based on some degree of similarity. No relationships among 3D objects can be easily answered. Also, no temporal aspect of 3D objects and metadata about them is examined.

- Nowadays, there exist some standardized 3D object description languages such as the Virtual Reality Modeling Language (VRML) [22] and the Extensible 3D (X3D) [23] standards. Also, the significance of the use of metadata in handling complex data is realized for multimedia, geographic data, and other resources. Indeed, MPEG-7 has been invented for audio/visual data, not including 3D objects. However, there is no specific metadata standard about 3D objects. Without any standard, each group of applications specifies its own metadata to deal with 3D objects at the semantic level. This makes 3D object management by means of metadata application-dependent. Thus, the sharability and reusability of 3D objects are reduced.
- Existing works related to metadata about 3D objects have some shortages. Firstly, their metadata extraction process is almost manually carried out or not considered. Secondly, there is little consideration on operations on metadata about 3D objects. Front-end applications that require metadata about 3D objects have to invent their own metadata consumption methods. Thirdly, no temporal aspect of metadata about 3D objects is taken into account as the time-varying of 3D objects takes place.
- As of this moment, the temporal aspect of 3D graphical data has been considered with continuous changes by means of abstract data types in the 3D space with the time dimension. Nevertheless, 3D object representations are of high complexity and manipulations on 3D objects are very limited as compared to the investigation of spatio-temporal databases in the 2D space.
- For the time-varying of properties, relationships, and/or 3D objects themselves, data timestamping is taken into consideration at both tuple/object and attribute levels. From this temporal aspect, we figure out some difficulties in dealing with temporal data at the attribute level as follows.
    - Existing SQL standards lack temporal supports, especially the one at the attribute level. This fact leads to no inherent temporal support for database users from any DBMS using SQL. With a non-temporal database language, it is a great burden for database users to handle temporal data all on their own. Also, temporal data management will vary from user to user and from application to application. In case of working together, different database users need to have an agreement on their temporal data representations and manipulations.

- Strong support for attribute history orientation is hardly obtained with the tuple/object timestamping scheme that has been researched in many works. In those works, an attribute history can not be treated as the whole one without a time invariant key for history identification. Also, the non-temporal and temporal information of a single entity will be split into several tuples/rows and tables with the tuple timestamping scheme. This leads to an unnatural temporal data representation with so-called horizontal and vertical temporal anomalies.
- In other research works, only some temporal attribute operations are supported for history orientation. Most of them focus on temporal data representation, algebraic operations, or a temporal extension to OQL mainly for temporal data querying. Works on schema definitions, schema evolutions, integrity constraints, and temporal modifications related to attribute histories in the attribute timestamping scheme still need to be done. Since other operations that are not supported as if built-in ones have to be manually implemented by database users using a non-temporal database language, temporal data management at the attribute level is limited.
- The non-procedural property of the standardized SQL language is very well-known for database users. It enables users to concentrate on what they want, rather than how they have to do to achieve what they want. Nevertheless, no temporal object relational SQL language exists with the attribute timestamping scheme for temporal database schema definitions, temporal database schema evolution definitions, the specification of integrity constraints, and temporal data manipulations. Therefore, a temporal extension to the standardized SQL language at the attribute level will significantly facilitate temporal data management in many aspects from definition (schema, schema evolution, and constraint) to manipulation (query, insertion, deletion, and update) like the non-temporal standardized SQL language has supported non-temporal data management up to now.

1.2 Contribution

In this research, a temporal object relational database system is originally proposed for 3D objects whose properties, relationships, and/or themselves are supported with valid time. The system will significantly facilitate the handling of both 3D graphical data and metadata parts of 3D objects along time.

- The temporal object relational data model is proposed together with a novel temporal object relational SQL language.
  - The temporal object relational data representation is defined with the attribute timestamping scheme. It is intuitive and expressive to naturally gather non-temporal and temporal attributes of the same entity type.
  - The language is upward compatible with the non-temporal object relational SQL language. It includes all main functions of a non-procedural database language for temporal database schema definitions, temporal database schema evolution definitions, temporal integrity constraint definitions, temporal data querying, and temporal data modifications. The support of temporal database schema evolutions enables the temporal enhancement on a non-temporal database.
  - As a temporal extension to the standardized object relational SQL language, the proposed language is non-procedural. By using this language for temporal data management, database users can specify what they want rather than how to achieve what they want. Also, the proposed language includes the intuitive expressive power from the temporal logic for temporal data querying. Hence, it allows database users to handle temporal complex data of any data type and issue temporal ad-hoc queries based on object relational technology with ease and intuition.
  - Also, the language possesses temporal upward compatibility to encourage both non-temporal and temporal database users to use the same means to facilitate data management regardless of the temporal aspect of data up to their requirements and abilities. Therefore, both non-temporal and temporal database users can work on the same database which is temporal to temporal database users and as if non-temporal to non-temporal database users.

- Furthermore, the language can be used in a temporal transparency environment through existing interfaces which are from an employed object relational database management system (ORDBMS); thus, well-known to database users. This feature is very helpful for temporal database application development on a non-temporal ORDBMS that is now temporal with our temporal extension.
- The temporal compatible object relational database system that we have achieved plays an important role of an underlying platform on which temporal database applications can be developed.

- The temporal object oriented NIAM conceptual schema model is introduced for conceptual data representation in respect of valid time. A temporal object oriented NIAM conceptual meta schema is defined on this temporal object oriented NIAM conceptual schema model to represent non-temporal/temporal metadata about 3D objects and their scenes at the abstract level. Such a conceptual meta schema is a common means for communication among users, graphics designers, front-end application developers, and domain experts. Moreover, the conceptual meta schema provides them with the overall understanding of 3D objects and their scenes via semantics rather than via graphics encoding and visualization.
- A procedure is determined to systematically and automatically obtain a temporal object relational database for temporal 3D objects from a 3D graphical data source that is true in reality for some period of time.
  - Firstly, based on our temporal object oriented NIAM conceptual schema model, a temporal conceptual meta schema is shaped from the input 3D graphical data source to represent metadata in respect of valid time about temporal 3D objects at the abstract level. The derivation of the temporal conceptual meta schema is application-independent and can be automatically carried out.
  - Secondly, based on our temporal object relational data model, a corresponding temporal meta database schema is systematically and automatically obtained from the corresponding temporal conceptual meta schema.
  - Thirdly, extracted from the 3D graphical data source, metadata about temporal 3D objects can be populated into the metadata part of the temporal object relational database while graphical data of temporal 3D objects into the

graphical data part of the temporal object relational database. Other metadata about temporal 3D objects that are not explicitly extracted from the input 3D graphical data source can be additionally populated into the metadata part of the temporal object relational database for those temporal 3D objects.

- The querying of 3D objects over the time is enabled in many aspects: graphical, semantics, and temporal by means of the standardized object relational SQL language extended with our temporal and 3D data supports. Due to the maturity and popularity of the SQL language, various front-end applications can conveniently have access to temporal 3D objects in a standard way.

1.3 Assumption

The research is conducted with several following assumptions.

- For the handling of the graphical data part of 3D objects, it is obvious that the handling of the graphical data part of 3D objects relies on the specific 3D graphical data format of the 3D graphical data source of 3D objects. So, the 3D object description language is chosen to be the standardized Virtual Reality Modeling Language (VRML) in our work. This implies that a 3D graphical data source fed into our system is formatted in VRML.
- For the handling of the metadata part of 3D objects, metadata will be explicitly extracted from a 3D graphical data source with the assumption that the scene graph of the source is composed of 3D objects of interest, not of graphical nodes. That is, 3D object recognition is excluded from this research and the handling of the metadata part of 3D objects is independent of the 3D object format. However, in combination with the previous assumption, an automatic data population process is developed for a 3D graphical data source formatted in VRML. In order to automate the process, it is supposed that each 3D object in a scene is generated from some 3D pattern of a known library also formatted in VRML. Each pattern is associated with a type name of a semantic type which is meaningful to humans and is a collection of 3D objects of interest in the Universe of Discourse. The library can be established by graphics designers and domain experts for 3D patterns and for their type names, respectively.

- For the handling of the temporal aspect of 3D objects, both tuple/object and attribute timestamping schemes are utilized. Nevertheless, the main concentration of the proposed temporal object relational SQL language is given to the attribute timestamping scheme with temporal upward compatibility, sequenced cases, and non-sequenced cases. Differently, the tuple/object timestamping scheme is only supported with non-sequenced cases where the timestamping attributes are equally treated as other attributes of an object. Besides, discrete changes of the properties and relationships of 3D objects are supported in this research work. As for continuity, the continuous changes of 3D objects, e.g. their motions, can be accommodated with some properly small granularity although not considered.

1.4 Dissertation Organization

This dissertation contains fourteen chapters and two appendices. Apart from this chapter for an introduction, the rest of this dissertation is structured as follows.

An overview on the works related to ours is given in the next chapter Literature Review with consideration on the handling of 3D objects and the handling of temporal data in general. From the literature review, problem statements are addressed in chapter 3 to specify the scope as well as the goals of this research. Next, chapter 4 will briefly present the theoretical background of fields in which the research is involved. Chapter 5 goes into detail over the handling of 3D objects. The graphical, semantic, and temporal aspects of 3D objects are examined. As for an overall illustration of the work, a case study is then described in chapter 6. The next two chapters 7 and 8 are dedicated to the conceptual data representation and temporal object relational data representation where the temporal object oriented NIAM conceptual schema model is proposed in chapter 7, the temporal object relational data model and the temporal object relational SQL language in chapter 8. With the conceptual and logical temporal data representation, a temporal object relational database is defined for 3D objects in respect of valid time in chapter 9. After that, chapter 10 shows our proposed temporal data population process from a 3D graphical data source for both graphical data and metadata parts of 3D objects with valid time. Using the result from chapters 9 and 10, chapter 11 presents the querying of 3D objects with/without temporal aspect by using

the temporal object relational SQL language earlier proposed in chapter 8. The expected output of this research is presented in chapter 12. It is a temporal object relational database system for 3D objects with the valid time and 3D data supports. A short conclusion of the entire work is drawn in chapter 13. Finally, chapter 14 will point out some possible interesting topics for further research based on our achievements.

Along with the chapters introduced above, two appendices are added at the end of the dissertation for more details. Appendix A shows the user's manual of our extension to an employed ORDBMS for temporal and 3D data supports. Using this appendix A, the demonstration of our proposed temporal compatible object relational database system and temporal object relational SQL language can be carried out. As for appendix B, a prototype of the proposed temporal object relational database system for temporal 3D objects is included with its requirements, functionalities, user interfaces, and the SQL statements of the temporal metadata population process on the case study. This appendix B exemplifies the temporal object relational database system for temporal 3D objects that is obtained from our research work.

Chapter 2

Literature Review

In this chapter, a literature review is presented on the handling of 3D objects. For the temporal aspects of 3D objects and metadata about them, an overview on the handling of temporal data is also included.

2.1 The Handling of 3D Objects

This subsection presents an overall review on the handling of 3D objects in many different aspects. [24, 25, 26, 27, 28, 29, 30, 31, 32, 33] consider only raw data. Among them, [24, 25, 26, 27, 28, 29] aim at temporal 2D/3D data and [30, 31, 32, 33] do not examine any temporal aspect of 3D data. [34, 35, 36, 37] using content-based search for dealing with 3D graphical data are also reviewed. Differently, [4, 5, 6, 7, 8, 9, 10, 11, 38, 39] handle 3D objects via their semantic information, called metadata. More general, [12, 13] handle not only metadata about 3D objects but also their graphical information.

[24, 25, 26, 27, 28, 29] are among works on spatio-temporal databases. [24, 25, 26, 27, 28] add the time dimension into spatial data by making use of abstract data types. Only temporal 2D spatial data is handled in [24, 25, 28, 29] while temporal 3D data in [26, 27]. Even though the investigation of 2D data can be extended to 3D data in principle, operations on 3D data must be a matter that needs to be further explored. [26] details a physical representation of 3D spatial data and deals with continuous changes of 3D geometries along time. Their work shows how complex the management of temporal 3D data is. Also, no operation on their 4D data types has been investigated. [27] takes VRML source files as an input in a manual data population process. This would be a tedious and heavy task for users as input scene graphs get larger. Besides, only 3D objects with simple shapes such as boxes, cones, cylinders, and spheres are supported. No support for 3D objects with complex shapes might restrict the applicability of their work because 3D objects might be captured from real objects that can be of any unanticipated form. Although the temporal aspect of 3D objects is

considered, no temporal relationships such as before, during, starts, overlaps, finishes, etc. among objects along the time axis can be obtained.

[30, 31, 32, 33] focus on a general solution to 3D data representation, not counting the temporal aspect of 3D data. [32] provides a formal definition of 3D primitives at the abstract level by means of mathematics with no implementation aspects. In order to introduce these 3D primitives to real world usage, their implementation needs more consideration. In contrast, the work in [30, 31, 33] handles 3D geometries at the implementation level with the spatial support of an ORDBMS. Due to the complexity of a 3D object in general, their representation of a 3D object requires deep knowledge of its shapes. In addition, the operations on 3D geometries are very limited from the current support of an ORDBMS although 3D data representation is allowed with DBMS-supplied spatial data types. Particularly, there are few functions of the checking of topological relationships among 3D objects and the z axis is omitted.

[34, 35, 36, 37] provide content-based search over 3D graphical data. [34] proposes a content-based search mechanism over a collection of 3D objects formatted in VRML. [35] introduces algorithms for 3D object recognition and segmentation in cluttered scenes which are obtained from range images. [36] gives us an analysis on 3D model descriptions and retrieval methods. A common problem resolved with content-based search is to fine a list of 3D objects from a given collection similar to an input object in a query based on some similarity measurement like [34, 36] or to fine which objects in a given library appear in a scene like [35]. The solution to this content-based search problem is investigated for individual objects based on some degree of matching similarity. It is noted that 3D objects in such a search in [34, 36] are simply gathered together in an arbitrary manner. They are defined in their own local coordinate system. Thus, spatial relationships among these 3D objects do not exist. As for [35], spatial relationships among 3D objects in a scene exist. However, they are not taken into account in [35], which mainly concentrates on 3D object recognition and segmentation in cluttered scenes. As a further investigation in 3D object retrieval, [37] introduces their consideration on semantic aspects of 3D objects. [37] develops a multimedia asset management system that allows associating metadata with physical entities, performing 3D content-based multimedia search, and enabling relevance feedback methods. Like

other works with a content-based search above, physical entities are individually handled. They have no spatial relationships with each other. Therefore, only simple queries with object names, types, and/or binary operators in the form of "field_name operator field_value" are supported together with keyword type and 3D content-based searches. The result of a query is also all stored objects that have 3D representations similar to the given object in the query. The relevance feedback from the user is expected in [37] for more relevant results. Generally speaking, relationships and the temporal aspect of 3D objects as well as non-presentable information about 3D objects cannot be reached in these works, which employ a content-based search.

Apart from content-based searching for 3D objects, metadata surrounding them plays an important role in the handling of these 3D objects as indicated in several current standards. The Dublin Core metadata standard [3] defines a vocabulary of fifteen elements for use in resource description. These elements are contributor, coverage, creator, date, description, format, identifier, language, publisher, relation, rights, source, subject, title, and type. As for geographic information, the ISO standard [2] accompanies the standardized spatial schema [40] to describe digital geographic data. The MPEG-7 standard [1] defines classical archival-oriented and perceptual descriptions about audio/visual data contents as well as additional information for organizing, managing, and accessing the contents in terms of low-level and high-level descriptors. However, these metadata standards do not take 3D objects into account. Also, they neither standardize a metadata extraction process from data sources nor specify metadata consumption issues. Nevertheless, these aforementioned standards set up a foundation from which more application-specific metadata needs can be derived. They figure out the significance of metadata in complex data management. As a marked point, there is no specific metadata standard for handling 3D graphical objects at the semantic data level as of this moment. Indeed, [4, 5, 6, 7, 8, 9, 10, 11, 38, 39] concentrate on the need of semantic information on 3D objects but with no temporal aspect of 3D objects and metadata about them. We examine these related works according to their foundation for the format of metadata about 3D objects as follows.

a. The MPEG-7 standard

[5, 6, 7] make use of the MPEG-7 standard. [5] establishes the links between visual, geometric, and semantic entities to the input X3D fragments. The annotation process seems to be manual. The manipulation over the achieved MPEG-7 semantic description has not yet been defined in [5]. As an extension of [5], [6] determines SQL-like queries against semantic instances. These queries are then intended to be translated into XQuery expressions because semantic information is organized in XML documents according to the MPEG-7 standard. For such a translation, a set of template queries is planned to be built. Realized in [6], this approach to querying semantic annotations on 3D objects is more restrictive than a free query approach. Similarly, [7] uses MPEG-7 to store semantic graphs associated with their scenes. A semantic entity is manually selected to describe some 3D object at some level of detail and of abstraction. Besides, [7] does not mention any manipulation on MPEG-7 descriptions.

b. Ontologies

In addition to the employment of MPEG-7, [8, 9] are based on ontology modeling for the semantics of 3D objects. Ontologies in [8] are the primitives and concepts that are used to build 3D objects and scenes. They are structured in Ontology Web Language (OWL) classes. Then, [8] organizes scene metadata in their specialized file format, called I3DVPproto, for the incorporation of semantics into 3D scenes. The access-limited logic (ALL) language is used to have access to metadata. This metadata organization is somewhat inconvenient to front-end application developers as they have to be trained with a new format. Their metadata achievement is manually carried out in [8]. Also using the notion of ontology, [9] uniquely identifies each virtual object on the Internet corresponding to an object type according to an ontology. Bounding boxes and spatial containment relationships among objects are able to be extracted as well. Their so-called spatial containment relations are in fact object compositions. The process of metadata extraction from a scene specification in [9] is partially automatic. Moreover, [9] does not discuss their global knowledge base, which contains annotated information: virtual worlds with their semantic information.

c. The Resource Description Framework (RDF) standard [41]

By employing RDF, [10] defines a number of RDF schema vocabularies each of which defines terms for a particular set of concepts surrounding the content of a virtual environment. The attachment of a 3D object (shape) to an appropriate RDF resource seems to be manually made by the graphics designer. Their semantic information must be organized in XML documents as RDF is XML-based. As for [11], the notions of a semantic object and a semantic zone are defined. A semantic object is used to represent one or more geometric objects. It has a specific meaning shared by the author and the final user of the 3D environment. Relations among semantic objects are also specified such as containedBy, sharedBy, and boundedBy. A semantic zone is used to represent spaces (zones) of a 3D environment. These semantic objects and zones are organized in a hierarchical structure and included into an X3D file via MetadataSet nodes together with 3D geometric objects. Apart from semantic information stored in X3D files, [11] uses RDF Schema files to capture the relations between the classes of semantic objects for some specific domain. With such an organization, manipulations on semantic information must directly refer to X3D and RDFS files. Nevertheless, no facility is introduced in [10, 11] for front-end applications to manipulate metadata.

d. The geographical metadata ISO standard for 3D city models

For 3D city models, [38] discusses metadata in regard to the geographic metadata standard [2]. After an analysis on the applicability of the metadata standard, [38] highlights the needs of a 3D spatial data infrastructure as well as of metadata catalogues particularly for 3D city models. However, no operation on metadata is analyzed towards front-end applications once metadata is available.

e. Semantic nets

[39] uses semantic nets for knowledge representation of a virtual world. Due to the nature of semantic nets, semantic information and their types are not separated in such a way that the fast growing of the net that might take place will have an impact on access to semantic information with graph traversing. Only access around instances (e.g. to ask if an instance is a kit-part) or relations (e.g. inst-of) of a semantic net is allowed. Thus, their semantic information consumption is confined.

Different from the related works which have been reviewed above, [12, 13] consider both graphical information and metadata about 3D objects. They propose a system with the 3D graphical data model called 3DGML to allow the semantics of 3D objects to be incorporated into a 3D scene. The semantic information captured includes component objects, spatial relationships about direction, and composition hierarchies of objects. In their system, VRML input files will be converted into 3DGML documents that are XML-based files containing both graphical and such semantic information. After that, these XML-based files are parsed and stored in a relational database. The system provides end-users with a Web-based interface to issues query-by-example style queries. Queries are enabled with only spatial relationships about direction such as left, right, above, below, near, and far. Although descriptive information about each individual object is introduced, where such information comes from and what types of 3D objects in a scene are not explained. Also, query results are entire scenes. No way is introduced to have access to particular parts of a scene. No topological relationships and temporal aspect of 3D objects and scenes are investigated as well.

2.2 The Handling of Temporal Data

In addition to the literature review on the handling of 3D objects, we pay attention to the handling of temporal data in general. This consideration will be essentially applied to the handling of the temporal aspect of 3D objects in particular.

Firstly, the data timestamping schemes are discussed. The timestamping of time-varying data can be done with both tuple/object and attribute timestamping schemes. The former one is simpler mostly with first normal form (1NF) relations in the relational data model [14] while the later is more complex, but more expressive, more natural, and closer to user thought process with less data redundancy. This is because all time-varying data of an object or attribute of interest are modeled together in a single tuple/object [42]. Furthermore, the former one might produce temporal data anomalies [43], namely vertical and horizontal, due to the splitting of the data of a single modeled entity/object into several tuples and in several relations. To overcome such unexpected anomalies, the latter shifts data timestamping to the finer attribute level. This leads to the concepts of a temporal attribute and an attribute history. This is one of reasons for the

use of the nested relational [15], object oriented [16], and object relational [17] data models to naturally deal with temporal data at the attribute level. For the tuple timestamping scheme, the existing works [20, 28, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54] have been researched on many issues of a temporal database including temporal data representations, integrity constraints, querying, modifications, and implementations. In comparison with temporal database systems with the tuple timestamping scheme, temporal database systems with the attribute timestamping scheme are less in number. Indeed, [55] provides an overview on temporal database system implementations. Among these temporal database systems, only the TDBMS system associated with the works [42, 56] based on non-1NF relations with basic temporal relational algebra operations and restructuring operations PACK and UNPACK follows the attribute timestamping scheme while others follow the tuple timestamping scheme. With the availability of the nested relational, object, and object relational database systems to support non-atomic attributes, the number of temporal database systems that support the attribute timestamping scheme in the literature has gradually increased and a few recent system implementations can be found in [24, 57, 58, 59, 60]. In this literature review on the handling of temporal data, we will concentrate on a valid time support at the attribute level and will consider the existing works [24, 42, 43, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66] with attribute timestamping for a valid time support.

Secondly, a so-called built-in temporal support is examined in some existing SQL standards and commercial database management systems (DBMSs). As of this moment, there is no built-in temporal support in any existing SQL standard. Part 7 SQL/Temporal [67] has been withdrawn and not yet implemented in any existing commercial DBMS although introduced a long time ago. Commercial database vendors are now working on a transaction time support, e.g. Oracle with FlashDB [68] and MS SQL Sever with ImmortalDB [69]. To our best knowledge, valid time support is provided by Oracle Workspace Manager [70] and no valid time support from Informix, IBM DB2, and MS SQL Server. However, the valid time support from Oracle 10g/11g [71] follows the tuple timestamping scheme and provides temporal database users only with relationship checking operators between two periods (wm_overlaps, wm_contains, wm_meets, wm_equals, wm_lessthan, and wm_greaterthan) and set operators

(wm_intersection, wm_ldiff, wm_rdiff). In general, the implementation of a temporal DBMS that supports valid time at the attribute level is rarely available to database users except employing proprietary services. With no built-in temporal support, a non-temporal database language has to be employed for temporal data management. As proved in [52], it is very complicated and difficult for database users to handle temporal data all on their own, even more with the attribute timestamping scheme.

Thirdly, an overview on the related works [24, 42, 43, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66] with a valid time support at the attribute level is given for a comparison.

a. With the relational data model

A research work in [58] introduces a temporal database system which provides database users with attribute timestamping while the tuple timestamping scheme is used for their underlying implementation on top of a relational DBMS. Their temporal extension to SQL is simple with the AS ON clause for the specification of the date from which retrieved data has been valid and the WHEN clause for the specification of search conditions on temporal attributes. The WHEN clause allows temporal data selection while the WHERE clause is evaluated with the principle of temporal upward compatibility. There is no discussion about joins, integrity constraints, and modifications on temporal data. Also developed on a relational DBMS, [61] defines temporal counterparts of existing operators performing on temporal attributes. Since the data type of an attribute can not be anticipated, this approach can not generally overcome the time-varying of data. So, the proliferation of temporal operators might take place for each individual data type.

b. With the nested relational data model

[42, 43, 56, 63] provide a temporal algebra/calculus with no consideration on time granularity, upward compatibility, temporal upward compatibility, integrity constraints related to time-varying data except [43] defines homogeneous constraints. Also, there is a restriction on data types of time-varying data (integers, real numbers, and strings) supported in [42, 56]. In their works, the support of conditions on time-varying data is little, thus resulting in some degree of complexity, as time variables are allowed to be directly accessed. This feature can be considered to be the non-restrictiveness (non-

sequenced) feature mentioned in [45, 52]. Furthermore, the implementation is likely to be unavailable to users [43, 56, 63] or partial [42].

c. With the object relational data model

On the employment of an ORDBMS, [64, 65] use temporal versions of data types. However, their approach is not generalized because valid timestamps are attached to a particular data type such as NUMBER. Moreover, temporal data manipulations such as temporal methods/routines are missing. Similar to [61], [64, 65] might incur the proliferation of temporal versions of data types that are not anticipated and the proliferation of temporal operations on instances of these temporally enhanced data types as a side effect. Along with [64, 65], [59] implements a temporal DBMS on Oracle 10g/11g. Like [58], [59] uses tuple timestamping to handle timestamped attributes. Indeed, temporal data is presented as temporal views each of which is defined as a merge of an original snapshot table and a temporal table corresponding to an individual temporal attribute. Besides, their temporal data representation is defined with instant timestamps that might result in data redundancy as compared to temporal element timestamps and incur some semantics assumption to provide an interval-based validity. Like the valid time support in Oracle 10g/11g Workspace Manager, their temporal support includes only temporal relationship operators. Temporal upward compatibility and the details of their temporal views, temporal data querying, temporal data modifications, and integrity constraints are not discussed. In addition to [53, 64, 65], we are aware of a transaction time support of [72, 73] that is provided on an ORDBMS. The main differences between valid time and transaction time are pointed out as follows. Transaction time is system-supplied while valid time user-supplied according to the modeled world. No modification on transaction time is directly allowed database users while modifications are possible on valid time. Transaction time is not able to stretch the future while valid time can be supported towards the future. As far as we know, valid time and transaction time are orthogonal to each other. Despite our focus on a valid time support, our work can be nicely extended to cover transaction time. This can be done either from scratch or by combining a built-in transaction time support already supplied by an existing ORDBMS in order to become fully temporal, i.e. bi-temporal.

d. With the object oriented data model

On the use of the object oriented data model, [24, 57, 60] use the concept of a history while [62, 66] define a parameterized data type to lift a data type to a temporal one by adding a time dimension to that type. [57] has a restriction on data type support because their point-wise generalization is only made for the arithmetic operators on integers. In addition, [57] uses instants, coalesced intervals, and instant sequences for timestamping. Since the choice of such timestamps is not in the well-known closed-open format, [57] cannot provide continuity checking for the validity of a fact in reality. Indeed, [57] incurs so-called semantic assumptions which might have an impact on some degree of data precision in temporal data management. As for [62], time intervals are used for timestamping; so, data redundancy might appear. [62] does not discuss temporal joins while temporal data querying in [60] produces no history oriented preservation in the SELECT clause and does not perform temporal joins when two pieces of temporal data appear in the FROM clause. Instead, [60] provides a new operator "tstruct" for temporal joins. Like [61], [66] is limited by the support of temporal operators with the attachment of temporal aspects to comparison operators. Also, [66] has the temporal support for only integers and booleans. Furthermore, only [57, 62] provide temporal upward compatibility while others do not take this feature into account. Since based on the OQL language, temporal data modifications are not mentioned in [62, 66] or made through C++ programming language bindings in [24, 57, 60]. Such a temporal support in [24, 57, 60] will confine temporal database application development to a particular programming language, e.g. C++. Similarly, [24] is specific for spatial data types. [24] takes into account only historical containment constraints on temporal data while others have no discussion on integrity constraints related to time-varying attributes. For a temporal extension to ODL/OQL, the implementations introduced in the works [57, 60, 62] are carried out on top of an OODBMS while the implementation in [24] is from scratch. That is, a new interface is provided so that the temporal support can get built-in to users. So, users' effort might be required for temporal data management via their new interfaces if their temporal database languages are employed.

Chapter 3

Problem Statement

This chapter addresses our problem statements in the research scope for 3D graphical objects in regard to valid time with a query capability. Before going into detail, we summarize in Table 3.1 the literature review which has been given in chapter 2.

Table 3.1 Summarization on References

Consideration on	References
3D Graphical Data of 3D Objects	[26, 27, 30, 31, 32, 33, 34, 35, 36]
with Database Technology	[26, 27, 30, 31, 32, 33]
with Content-based Search	[34, 35, 36]
Metadata about 3D Objects	[4, 5, 6, 7, 8, 9, 10, 11, 38, 39]
Both	[12, 13, 37]

The first group of the related works [26, 27, 30, 31, 32, 33, 34, 35, 36] pays attention to only 3D graphical data. There are two sub groups for the handling of 3D graphical data. One sub group including [26, 27, 30, 31, 32, 33] takes advantage of database technology and the other one consisting of [34, 35, 36] supports content-based search over 3D graphical data. The second group of the related works [4, 5, 6, 7, 8, 9, 10, 11, 38, 39] takes into consideration semantic information, i.e. metadata, about 3D objects for the handling of those 3D objects. The third group of the related works [12, 13, 37] deals with both graphical data and metadata parts of 3D objects. From the literature review, we emphasize the fact that the handling of temporal 3D objects by means of temporal metadata is a promising topic while dealing with raw 3D graphical data is quite complicated, especially with little support for 3D data manipulations from existing DBMSs.

For the related works with content-based search, 3D objects are individually handled with no temporal aspect, not defined in a global coordinate system; thus have no spatial relationships with each other. However, spatial relationships among them can not be derived with the content-based search approach even if 3D objects are spatially gathered together to form a scene. The query (retrieval) of these 3D objects is based on matching similarity normally with a query-by-example approach.

As for the related works on metadata about 3D objects, the query capability in these works is limited. Spatial relationships among 3D objects are little supported. Aggregation of related 3D objects cannot be easily computed from resultant metadata. Especially, no temporal requirements can be expressed because no temporal aspect of 3D objects and metadata about them is investigated. None of the reviewed works takes into account the temporal aspect of metadata about 3D objects that are time-varying. The dynamic feature of 3D objects can not be reflected. This is an obvious shortage as changes about 3D objects are frequently made over time. Those changes lead to changes of corresponding metadata. For example, a new/existing object enters/exits a scene or an existing object in a scene changes its position.

About the related works that handle both graphical data and metadata parts of 3D objects, the query support to have access to 3D objects is restricted via some fixed query interface. Either single objects or entire scenes can be retrieved. There is no way to consider particular parts of a scene, topological relationships among 3D objects in a scene, and dynamic features of 3D objects along time. Especially, no temporal aspect is examined for both graphical data and metadata parts of 3D objects in those works.

In short, it is claimed that a temporal object relational database system for a large number of 3D objects has not yet existed for the handling of both 3D graphical data and metadata parts of 3D objects and their scenes over the time.

Chapter 4

Theoretical Background

In this chapter, we briefly present the theoretical background of the research scope where database technology is made the most of for the handling of a large number of 3D objects and their scenes along time.

4.1 Database

4.1.1 An Overview

In general, *a database* is *a collection of data and relationships*. In the context of information systems, the definition of a database is considered in a structured manner; that is, a database is based on some logical data model supported by a database management system (DBMS). This is a so-called database approach. Some existing logical data models are hierarchical, network, relational, nested relational, object oriented and object relational data models. In contrast, a database which is not based on any data model is meant to be all created, manipulated, and managed in application programs. This is a so-called traditional file processing approach.

The file processing approach is summarized as follows.

- Characteristics: not based on any data model, the database is simply a collection of data all handled (created, manipulated, and maintained) by application programs.
- Advantages
  - Files of data are useful for the storage of unstructured or very differently structured data.
  - Data are handled at a relatively low level without any overhead from any layer between data and application programs.
- Disadvantages
  - Less flexibility in data structure
  - Hard for redundancy control
  - Hard for multi-user support with concurrency control
  - Hard for security with access control

- High programming complexity without data-independence

Different from the file processing approach, the database approach is associated with some data model supported by a DBMS.

- Characteristics: the database is all handled (created, manipulated, and maintained) with a data model supported by a DBMS.
- Advantages: overcome the disadvantages faced in the file processing approach with facilities provided by the DBMS. Several facilities are concurrency control, backup and recovery control, query optimization, security issues, and so on.
- Disadvantages
  - Incur the overhead of the use of the DBMS
  - Not suitable for database applications which are simple with a small amount of data, related to static data (write once, read many) and/or with no multi-user access.

It is noted that the main motivation of the database approach over the file processing approach is data independence according to the three-schema ANSI/SPARC architecture [74] for database systems as shown in Figure 4.1.

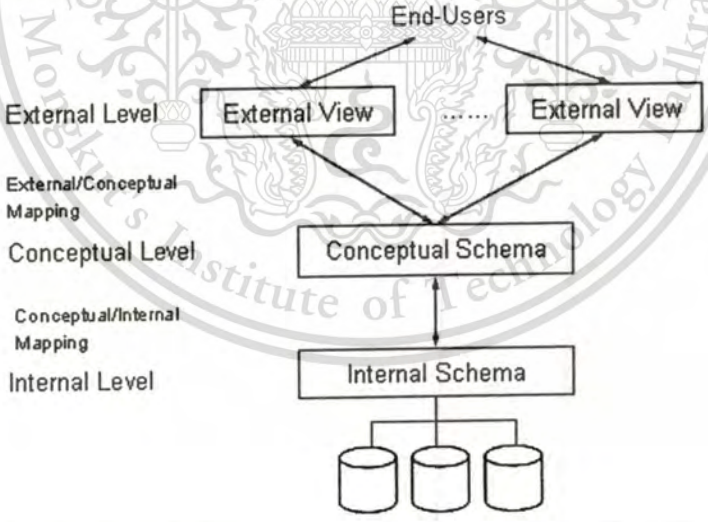


Figure 4.1 The ANSI/SPARC Architecture of a Database System

- The internal level (lowest) with an internal schema for the physical storage structure of the database,
- The conceptual level with a schema for the structure of the whole database based on some data model,

- The external (view) level (highest) with external schemas or user views.

Based on this three-schema architecture, data independence is the capacity of changing the schema at one level of a database system without having to change the schema at the next higher level. There are logical data independence and physical data independence.

- Logical data independence is the capacity of changing the conceptual schema without having to change external schemas; so not having to change application programs.
- Physical data independence is the capacity of changing the internal schema without having to change the conceptual schema; therefore, the external schemas need not to be changed as well.

4.1.2 Data Model

In this subsection, we distinguish between conceptual data models and logical data models. A logical data model is implemented by some DBMS while a conceptual data model is used to shape the structure of a database from users' data requirements at the abstract level. Different from a logical data model, a conceptual data model mainly aims at a concise description of the data requirements of the users and is rarely associated with a language for definitions and manipulations at this high level of abstraction.

4.1.2.1 Conceptual Data Model

Nowadays there are a few popular conceptual data models such as the Entity Relationship Model (ERM) [75], the object oriented conceptual data model with the Unified Modeling Language (UML) [76], and the NIAM (Nijssen's Information Analysis Methodology) conceptual data model [77], which is now known as the Object Role Model (ORM). A conceptual schema based on a conceptual data model includes detailed descriptions of the entity types, relationships, and constraints, not including the implementation details. These elements are expressed using the concepts provided by the conceptual data model at the high level of abstraction. They are usually easier to be understood as compared to the notions of a logical data model. So, a conceptual schema is used as a common means for communication among non-technical users, application developers, and domain experts. It is also used as a reference to ensure that

the user's requirements are met and that those requirements do not conflict with each other. Each conceptual data model should be proposed with a conceptual/logical transformation mapping algorithm to transform a corresponding conceptual schema to a database schema in some logical data model implemented by the employed DBMS. A brief overview on the popular conceptual data models is given in Table 4.1.

Table 4.1 An Overview on the Popular Conceptual Data Models

Conceptual Data Model	ERM	UML	NIAM/ORM
Set of objects of interest	Entity type	Class	Object type
Set of dependent objects	Weak entity type	-	-
Object attribute	Attribute	Attribute	-
Object identity	Key attribute	OID (implicit)	Identifier
Relationships	Relationship type	Relationship type	Relationship type
Constraints	Fewer	OCL expressions	Richer
Object behaviors	No	Class method	No
Theoretical Foundation	No	Object oriented paradigm	Predicate logic and linguistic
Relational mapping algorithm	1NF relations	1NF relations	5NF relations
Validation technique	No	No	Yes with sample populations
Conceptual Schema Design Procedure	No	No	Yes

Among the popular conceptual data models above, we employ the NIAM conceptual schema model as a base conceptual data model for our handling of 3D objects along the time line at the abstract level. As introduced in [77], the NIAM conceptual schema model follows the fact-based approach. The unit of data representation in the NIAM model is a fact type. A fact is an elementary proposition about a relationship among objects in the universe of discourse (UoD). Objects can be classified as either lexical or non-lexical objects, which are of label types or entity types, respectively. The NIAM model is based on logic predicates and linguistics with the use of a natural language. Natural language sentences are expressed in a fixed structure which can be interpreted in a single and unambiguous way in order to express elementary facts. This feature allows a NIAM conceptual schema that comprises entity types, label types, fact types, reference types, and integrity constraints to be transformed into fifth normal form relations based on the relational data model [14].

4.1.2.2 Logical Data Model

As defined in [78], a logical data model is a combination of three components.

- A collection of data structure types (the building blocks of any database that conforms to the model)
- A collection of operators or inference rules, which can be applied to any valid instances of the data types listed above, to retrieve or derive data from any parts of those structures in any combination desired
- A collection of general integrity rules, which implicitly or explicitly define the set of consistent database states or changes of state or both – these rules may sometimes be expressed as insert-update-delete rules.

A data model is employed for data management with several following purposes.

- As a tool for specifying the kinds of data and data organization that are permissible in a specific database
- As a basis for developing a general design methodology for databases
- As a basis for coping with evolution of databases so as to have minimal logical impact on existing application programs and terminal activities. Data models have paved the way for the much clearer separation of semantic issues from implementation issues in programming languages.
- As a basis for the development of families of very high level languages for query and data manipulation. That means that the operators or inference rules should, however, provide a yardstick of manipulative and query power.
- As a focus for DBMS development
- As a vehicle for research in the behavioral properties of alternative data organizations.

An overview on the popular logical data models: relational [14], nested relational [15], object relational [17, 18], and object [16] data models is given in Table 4.2.

The choice of a base logical data model on which our proposed temporal database system is built is made between the object oriented data model and object relational data model for the natural notion of "object" for 3D objects in their scenes. Moreover,

based on the capability of handling complex queries on complex data as shown in Table 4.3 [18], the object relational data model is the most relevant.

Table 4.2 An Overview on the Popular Logical Data Models

Data Model	Key Construct	Identity	Referential Constraint	Language
Relational	Relation with atomic attributes	Primary key (attribute values)	Foreign key (attribute values)	Relational algebra, tuple relational calculus, SQL:89, SQL:92
Nested Relational	Nested relation with nested relation attributes	Primary key (attribute values)	Foreign key (attribute values)	Nested relational algebra with nest/unnest operations
Object Relational	Relation/un-encapsulated object with atomic/non-atomic attributes	Primary key (attribute values) / OID (ROWID, REFC) (system-generated)	Foreign key (attribute values)/logical pointer REF (system-generated)	SQL:3, SQL:99, SQL:2003, OQL
Object	Fully encapsulated object with atomic/non-atomic attributes	OID (system-generated)	Logical pointer REF (system-generated)	Methods calling

Table 4.3 Classification of data management systems [18]

	Simple Data	Complex Data
Simple Queries	File System	Object DBMS
Complex (Ad-hoc) Queries	Relational DBMS	Object Relational DBMS

4.1.3 An Object Relational Database

From the determination above, this subsection will give us more details of the object relational data model on which object relational databases are defined. Object relational databases try to fill the gap between the relational and object oriented databases [79]. The object relational data model is achieved from the fusion of the object oriented and relational data models.

An *object relational database DB* is a collection of first normal form (1NF) or non-1NF *relations* with either atomic or non-atomic attributes from the relational point of view or a collection of *objects with non-encapsulated attributes* from the object point of view. In terms of tables, an OR database *DB* is a collection of *table*s each of which is either a *1NF or non-1NF table* not derived from any structured type or a *typed table* derived from a *structured type* [17], respectively. A structured type is a user-defined type including a list of attribute definitions. An instance (i.e. object) of this type corresponds to a row (i.e. tuple) in some typed table with attribute values that can be directly accessed without any restriction of encapsulation. An attribute value can be atomic or non-atomic. A non-atomic value can be an instance of a collection type or a user-

defined type. Different from a 1NF/non-1NF table, a typed table has an extra column called REFC [17]. The REFC column cannot be updated. Its value is determined during the insertion of a row into the typed table and used to uniquely distinguish the row from others. It can be system-generated or user-generated up to user's demand and the support of an underlying ORDBMS. Thus, the existence of this column can replace the concept of the primary key in the relational data model [14]. In addition, there exists a REF value referencing a row in a typed table. These REF values form a reference type of the structured type on which that typed table is defined. Hence, the notion of referencing can substitute the concept of the foreign key in the relational data model.

In this chapter, an OR database, a table, and a structured type are formally defined.

Let $ST = \{ST_1, ST_2, ..., ST_i, ..., ST_m\}$ be a finite collection of structured types. Each structured type ST_i for any i in 1..m contains a list of n attributes each of which A_j for any j in 1..n is of any AT_j type such as a built-in type (number, char, etc.), a DBMS-supplied type (spatial, text, etc.), a reference type (REF), a collection type (array, multiset), or a user-defined type.

Definition 4.1 (*Structured type, ST_i*). $ST_i = (A_1: AT_1, A_2: AT_2, ..., A_j: AT_j, ..., A_n: AT_n)$.

Definition 4.2 (*Table, ut_k*). $ut_k = \{ ..., (av_1, av_2, ..., av_j, ..., av_n), ...\}$.

A table ut_k which is a 1NF, non-1NF, or typed table is defined as a finite unordered collection of rows/tuples. If ut_k is a 1NF/non-1NF table not defined on any structured type, one row/tuple $(av_1, av_2, ..., av_j, ..., av_n)$ in ut_k belongs to the Cartesian product $D_1xD_2x...xD_jx...xD_n$ of n domains $D_1, D_2, ..., D_j, ...,$ and D_n. Each D_j domain for j in 1..n is a collection of either atomic or non-atomic permitted values which are all of a data type AT_j. In particular for any j in 1..n, the av_j value is of a data type AT_j, either atomic or non-atomic, and located at the column named A_j. If ut_k is a typed table defined on some structured type ST_i for any i in 1..m, one row/tuple $(av_1, av_2, ..., av_j, ..., av_n)$ in ut_k is regarded as an instance (object) of the ST_i type. For any j in 1..n, the value av_j at the column corresponding to the A_j attribute of the ST_i type is of a data type AT_j and can be either atomic or non-atomic. Also, each row in ut_k is associated with a value which uniquely identifies that row and is called an object identifier (OID) value. Apart from a list of attributes specified by database users, OID is an extra special attribute. Relationships among rows of one or many typed tables are represented by references whose values

are OID values of reference types. A reference referring to a row in ut_k is of the reference type REF (ST_i) in the SQL standards. OID is REFC in the OR SQL standard [17].

With the definitions of a structured type and a table above, an OR database *DB* is formally defined as follows.

Definition 4.3 (*Object relational database, DB*). $DB = \{ut_1, ut_2, \ldots, ut_k, \ldots, ut_z\}$.

The language accompanying the OR data model for manipulations on an OR database is the one with the ad-hoc capability of the relational database language and the extensibility of the object database language. Such languages are SQL:3, SQL:99, SQL:2003 [17, 80, 81], and OQL [16]. As introduced in [82], SQL:2003 is the standard inheriting all parts from SQL:1999 with the addition of a new part Part 14: SQL/XML (XML-related Specifications) and without the part Part 7: SQL/Temporal [67]. One of the most important features of these languages is the ability of the definition of user-defined types and operators performing on data of the new user-defined types [79, 80] for extensibility. Two varieties of extended datatypes which are expected to be supported in the object relational data model are distinct data types and opaque datatypes [18]. Distinct datatypes are user-defined types that share their internal representation with an existing type called source type, but which are considered to be a separate and incompatible type for most operations. Opaque datatypes are types whose precise definition is not documented and which are intended to be manipulated only using the documented interface, including a set of functions.

In practice, an OR database is supported by an ORDBMS for complex data and ad-hoc queries [18]. Operations such as selection, projection, Cartesian product, join, union, intersection, and difference are defined in the OR SQL language (SQL:3, SQL:99, SQL:2003) in the form of SELECT...FROM...[WHERE...] [GROUP BY...] [HAVING...] [ORDER BY...] statements for data querying. Modifications against tables are made by INSERT, DELETE, and UPDATE statements also specified in the OR SQL language. An INSERT statement is used to insert a new row into a table; a DELETE statement to remove one or many existing rows from a table; and an UPDATE statement to update existing attribute values of one or many rows in a table. The capability of the OR SQL language allows the handling of complex data types, e.g. 3D geometric types, by means of abstract data types, user-defined methods (functions/procedures), and

operators. Therefore, the employment of the OR data model for temporal complex data management is promising.

4.2 Temporal Database

As defined in [19], a temporal database is a database that supports some aspect of time, not counting user-defined time. Temporal aspects as summarized in Table 4.4 are valid time, transaction time, and user-defined time.

Table 4.4 An Overview on Temporal Aspects

Temporal Aspect	Valid time	Transaction time	User-defined time
Definition	Time when the fact is true in reality	Time when the fact is present in the database	Time un-interpreted and parallel to other data domains such as NUMBER, CHAR, etc.
Provided by	User	System	User
Modified by	User	System	User
Supported by	System	System	User
Special language for its semantics	Yes	Yes	No

- Valid time: the valid time of a fact is the time when the fact is true in the modeled world. A fact may have associated any number of instants and time intervals, with single instants and intervals being important special cases. Valid times are usually supplied by the user.
- Transaction time: a database fact is stored in a database at some point in time, and after it is stored, it is current until logically deleted. The transaction time of a database fact is the time when the fact is current in the database and may be retrieved. As a consequence, transaction times are generally not time instants, but have durations. Transaction times are consistent with the serialization order of the transactions. They cannot extend into the future. Also, as it is impossible to change the past, (past) transaction times cannot be changed. Transaction times may be implemented using transaction time commit times, and are system-generated and system-supplied. While valid times may only be associated with "facts", transaction times may be associated with any database object.
- User-defined time: user-defined time is an uninterpreted attribute domain of date and time. User-defined time is parallel to domains such as string and integer. Unlike

transaction time and valid time, user-defined time has no special query language support. It may be used for attributes such as "date of birth" and "hiring date".

A database without any temporal aspect is a non-temporal database where only latest (current) state of data is captured. A database supporting both valid time and transaction time is a bi-temporal database where all past, current, and future states of data are kept. Therefore, the main motivation of a temporal database over a non-temporal database is the capability of dealing with not only the current state but also the past and/or future states of data along the time line. This discrimination between a non-temporal database and a (bi-)temporal database is shown in the following Figure 4.2.

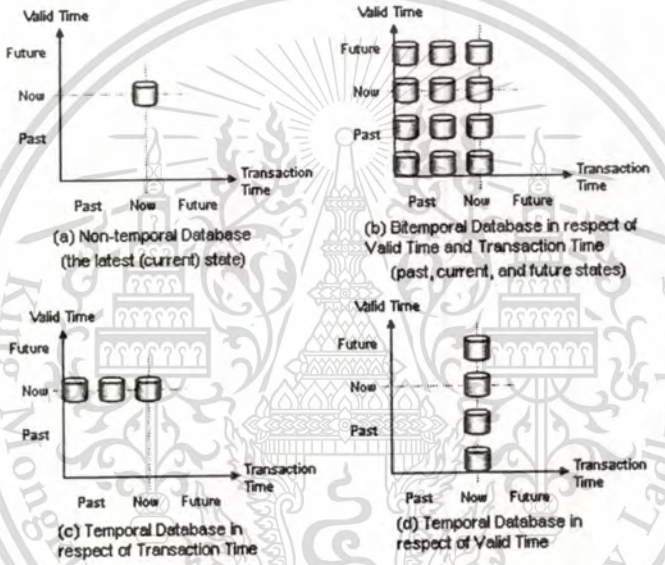


Figure 4.2 Non-temporal Databases and Temporal Databases

Table 4.5 An Overview on the Two Timestamping Schemes

Feature	Tupe-timestamping Scheme	Attribute-timestamping Scheme
Timestamp	Point (Instant), Interval (Period)	Point, Interval, Interval Set
Timestamped Level	Tuple/Object	Attribute
Base Data Model	Relational, Object Oriented, Object Relational	Nested Relational, Object Oriented, Object Relational
Advantages	Simple	Expressive, more natural representation, less redundancy
Disadvantages	- Ambiguous to identify time-varying data in a tuple/object - Unnatural representation with the splitting of data into many tuples and relations → so-called temporal anomalies: vertical and horizontal - Data redundancy	Complex with the non-atomicity of temporal attributes

As overviewed in Table 4.5, there are two fundamental approaches to the timestamping of time-varying data: tuple(object) and attribute. The choice of a timestamping scheme depends on the data model from which temporal data structures will be defined. The former is simpler to guarantee the first normal form relations of the relational data model whereas the latter is more complex; but more expressive, more natural and closer to user thought process with less data redundancy as all time-varying data of an object or attribute of interest are modeled together in a tuple (object) [42]. As determined in [20], the tuple(object)-timestamping scheme regards the timestamp as special attribute(s) of a temporal relation scheme and hence is part of every tuple while the attribute-timestamping scheme regards the timestamp as part of attributes and not of the whole tuple (object).

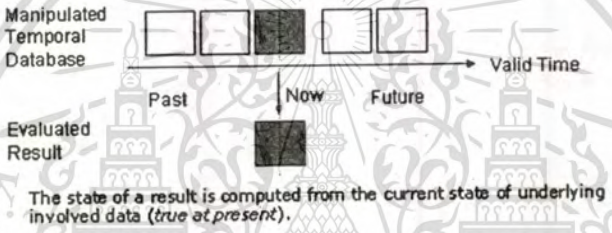


Figure 4.3 Current Data Manipulations

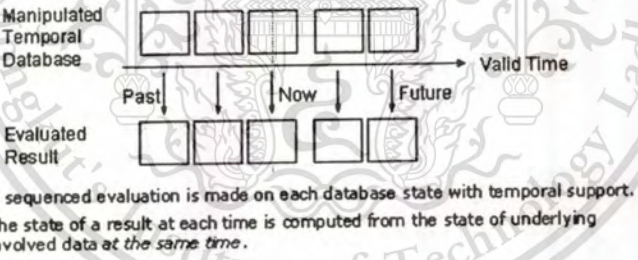


Figure 4.4 Sequenced Data Manipulations

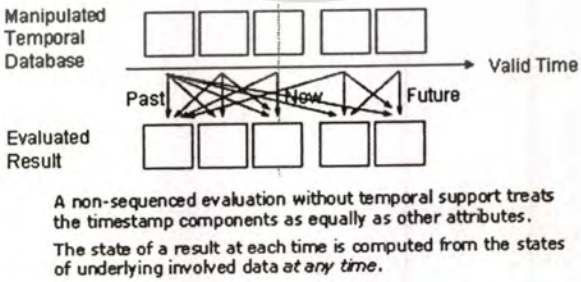


Figure 4.5 Non-sequenced Data Manipulations

Temporal database systems have been so far developed on the relational, nested relational, object relational, and object oriented data models. In those systems, temporal

database languages have been investigated as temporal extensions to relational algebra [20], nested relational algebra [42], QUEL [50], OQL [83], and SQL [48, 84, 85, 86]. According to the lesson in [52], a proposed temporal extension should at least support current manipulations, sequenced manipulations, and non-sequenced manipulations on temporal data. Current data manipulations in Figure 4.3 allow the state of a result to be computed from the current state of underlying involved data, i.e. data true at present. For sequenced data manipulations in Figure 4.4, a sequenced evaluation is made on each database with temporal support. The state of a result at each point in time is computed from the state of underlying involved data at the same time. In contrast to sequenced data manipulations, non-sequenced data manipulations in Figure 4.5 treats the timestamp components as equally as other non-temporal attributes, i.e. there is no temporal support for a non-sequenced evaluation. The state of a result at each point in time is computed from the states of underlying involved data at any time. Nevertheless, no temporal extension has been standardized or accepted to be included into some existing standard database language.

A particular branch of temporal databases is spatio-temporal databases [87] for the handling of both spatial and temporal aspects of data. Spatial characteristics of an object of interest in reality mainly include geometry and position. Others can be derived from geometry and/or position of the object. Once these spatial characteristics become time-varying, the object is spatio-temporal. This spatio-temporal kind of databases is of high interest due to the ubiquity of space and time. The support for spatio-temporal databases is normally done with an extension of abstract spatio-temporal data types and user-defined operations. Among operations on spatio-temporal data, operators for checking spatial relationships in respect of time are the most interesting. Those spatial relationships are about measurement, direction, and topology. A typical spatio-temporal database system has been proposed in [24].

4.3 3D Graphical Data Descriptions of 3D Objects

4.3.1 3D Object and Scene Construction

As previously mentioned in chapter 1, 3D objects can be acquired from real objects using some specialized techniques such as 3D laser scanning, photographic virtual

reality, etc. Also, many existing 3D authoring tools (Studio 3D Max [88], trueSpace [89], Internet Space Builder [90], AutoCAD [88], etc.) and 3D programming toolkits (Direct3D [91], OpenGL [92], Java3D [93], etc.) can help to create 3D objects and their scenes. Besides, there exist some standards (Virtual Reality Modeling Language (VRML) [22] and X3D [23]) for describing 3D objects and their scenes in a textual format. Each 3D object and scene construction method is suitable for each group of users and each class of 3D objects and scenes that are going to be constructed. Among these methods, we pay attention to the use of the VRML/X3D language for 3D object and scene descriptions. This is because the language is standardized and supported as input/output encodings by many available tools and toolkits. Moreover, the VRML/X3D language is an open standard that can allow lower costs, easier reusability of 3D data content, and easier integration with existing and future contents and applications.

4.3.2 3D Object Description Languages

The *Virtual Reality Modeling Language (VRML)* is an open ISO standard for describing 3D objects and 3D virtual environments, i.e. scenes composed of 3D objects. It is still the most known and used technology for building 3D data content as of this moment. 3D data content kept in text files comprises 3D objects and their scene by means of a hierarchical scene graph that is a directed acylic graph. A scene graph contains nodes. VRML defines 54 different node types, including geometry primitives, appearance properties, sound and video, and node types for animation and interactivity. Each node has some properties in fields. The language defines 20 different field types that can be used to handle data of different types from single integers to arrays of 3D rotations. The language also provides the programmers with a mechanism to define new nodes through a statement called Proto. In addition, a message-passing mechanism is defined in the language to allow nodes in the scene graph to communicate with each other by sending events for animation and interaction. More complex behaviors can be included in a VRML document using Script nodes that allow VRML nodes to be managed with programs written in Java or JavaScript.

Recently, a new ISO standard, called *eXtensible 3D Graphics (X3D)*, has been proposed as a successor of the VRML language. The X3D language inherits most of the

design choices and technical features of the VRML language. As a result, it is mostly backward-compatible, i.e. many VRML documents require only minimal changes for translation to X3D ones. First, it adds new nodes and capabilities, mostly to support advances in 3D graphics techniques and hardware, such as programmable shading and multi-texturing. Second, it introduces additional data encoding formats. More specifically, it is possible to represent, store, and transmit X3D content using a VRML-like textual encoding, an XML-based textual encoding, and a binary encoding.

Both VRML and X3D languages are managed by the Web3D Consortium [94], and results from the effort of several organizations, researchers, and developers worldwide.

Apart from the standardized VRML/X3D language, a few extensions to the VRML/X3D language have been achieved so far such as *GeoVRML* [95] for spatial aspects and *VRML-History* [96, 97] for historical aspects. The invention of the GeoVRML language stems from the requirements of representing and visualizing geographical data by using a standard VRML browser. [95] provides the support for spatial reference frames, precision, scalability, metadata, animation, introspection, and navigation with a collection of new node types: GeoCoordinate, GeoElevationGrid, GeoLocation, GeoLOD, GeoMetadata, GeoOrigin, GeoPositionInterpolator, GeoTouchSensor, GeoViewpoint, and InlineLoadControl. Among limitations realized in [95], the one that GeoVRML 1.1 does not address issues of time is of our interest. Another one is about the coupling of the geographic data and the human-readable metadata surrounding it with GeoMetadata nodes. This mechanism makes the entire scene graph traversed whenever metadata is required. For historical (temporal) aspects, the VRML language is extended in [96, 97] with a temporal support for the integration of valid time and time variant objects. The capability of the VRML History language is representing time-variant and time-invariant 3D and multimedia objects with different time dependent representations where each representation is valid for a specific period of time. Once valid time of each object in the scene is captured, the scene graph becomes a temporal scene graph. Both time variant and time invariant objects are included in this temporal scene graph hierarchy. Similar to GeoVRML, VRML History defines a few new node types with regard to valid time through EXTERNPROTO statements. These new node types are Entity, History, HistoryTexture, HistoryGeometry, NavigationInfo,

ProximitySensor, TimeInterpolator, ValidPeriod, ValidTimeSensor, and ViewPoint. Different from GeoVRML, VRML History does not discuss metadata issues. In common, both extensions in [95, 97] support only the 3D data representation and visualization in spatial and temporal aspects. With the use of GeoVRML and VRML History, 3D spatial and temporal data are captured and stored in VRML text files. Access to the 3D spatial and temporal data and their metadata must be placed at the application level as VRML text files need to be properly examined in application programs.

In addition to the VRML/X3D language and its extensions, we are aware of the *Geographic Markup Language (GML 3.0)* in [98] for geographic data encoded in the Extensible Markup Language (XML) and the *CityGML* language [99] specific for 3D city models such as Site, Building, Transportation Objects, Vegetation, Water Bodies, Furniture, and so on. GML 3.0 mainly focuses on the data representation of geometric and topological features. There is no consideration on other properties of spatial objects such as semantics and appearance aspects. Based on GML 3.0, CityGML represents 3D geometry, 3D topology, semantics, and appearance in five discrete scales (levels of detail, LOD): LOD0, LOD1, LOD2, LOD3, and LOD4. LOD0 is with 2.5D digital terrain models, LOD1 with block models without roof structures, LOD2 with models with textured, differentiated roof structures, LOD3 with detailed architecture models, and LOD4 with "walkable" architecture models. Similar to the VRML/X3D language, CityGML is intended to be a source structure for visualization processes. Different from the VRML/X3D language, CityGML includes the semantics of 3D city models into the visualization for interpretation processes.

4.4 Metadata Issues

To our best knowledge, metadata plays an important role in the handling of complex data in general and of 3D objects in particular. Nevertheless, there is no metadata standard for the handling of 3D objects in contrast to audio/visual data with the MPEG-7 standard [1] and geographic information with the ISO standard [2] as mentioned in chapter 2. Without any standard, the investigation of metadata surrounding 3D objects varies from proposal to proposal. Thus, there is no assessment on the properness and completeness of metadata about 3D objects that has been determined

in existing works up to date.

For multimedia (audio/visual) data, the MPEG-7 standard [1], which is an XML metadata standard, includes four key constructs: Descriptor, Description Scheme, Description Definition Language, and Systems Tools. A Descriptor is a representation of a Feature that is a distinctive characteristic of the multimedia data that signifies something to somebody. A Descriptor defines the syntax and the semantics of the Feature representation. A Description Scheme is the structure and semantics of the relationships between its components, which may be both Descriptors and Description Schemes. The Description Definition Language is a language that allows the creation of new Description Schemes and Descriptors. It also allows the extension and modification of existing Description Schemes. Systems Tools are tools to support multiplexing of descriptions, synchronization of descriptions with content, delivery mechanisms, and coded representations for efficient storage and transmission and the management and protection of intellectual property in MPEG-7 Descriptions. With these constructs, MPEG-7 supports three types of XML-based descriptions: classical archival-oriented descriptions (content's creation and production processes, the use of the content, the storage and encoding formats of the content), innovative perceptual descriptions (the content's spatial, temporal, or spatio-temporal structures, low-level features in the content related to colors, textures, etc., and semantic information related to the reality captured by the content such as objects, events, and their interactions), and additional information for organizing, managing, and accessing the content. However, neither automatic nor semi-automatic feature extraction algorithms are inside the scope of the MPEG-7 standard. Also, the MPEG-7 standard does not specify the consumption of MPEG-7 Descriptions.

For geographic data, the ISO standard [2] provides a structure for describing digital geographic data by supplying metadata elements, defining a schema, and establishing a common set of metadata terminology, definitions, and extension procedures. The core metadata elements (mandatory and recommended optional) required for describing a dataset are listed as follows: dataset title, dataset reference date, dataset responsible party, geographic location of the data set (by four coordinates

or by geographic identifier), dataset language, dataset character set, dataset topic category, spatial resolution of the dataset, abstract describing the dataset, distribution format, additional extend information for the dataset, spatial representation type, reference system, lineage, online resource, metadata file identifier, metadata standard name, metadata standard version, metadata language, metadata character set, metadata point of contact, metadata date stamp. Unlike the MPEG-7 standard, this ISO metadata standard does not support content-based metadata.

For a resource in general and for electronic documents in particular, the Dublin Core metadata standard [3] defines a core of fifteen elements: contributor, coverage, creator, date, description, format, identifier, language, publisher, relation, rights, source, subject, title, and type. Similar to the ISO metadata standard specifically for geographic data, the Dublin Core metadata standard does not support content-based metadata.

For web resources, the Resource Description Framework (RDF) standard [41] nowadays gets more popular. One of the capabilities of the RDF standard is representing metadata about web resources as a directed labeled graph along with an XML-based syntax. A resource can be thought of as anything in UoD that is referred to by a single Uniform Resource Identifier (URI) reference. RDF itself is specified in an XML Schema called the RDF Schema (RDFS) including rdfs:Resource, rdfs:Class, rdfs:Literal, rdfs:Datatype, rdfs:Container, rdfs:ContainerMembershipProperty, rdfs:Member, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:Comment, rdfs:Label, rdfs:Range, rdfs:Domain, rdfs:isDefinedBy, and rdfs:seeAlso. Queries on RDF documents are supported with the SPARQL query language with graph pattern matching. One of the features of the RDF standard as compared to other standards above is that metadata may be incorporated with the data in the same RDF document. This will make an RDF graph larger and the RDF graph has to be entirely loaded and traversed for each query against the data as well as its metadata. As of this moment, the RDF standard has not yet supported any temporal aspect. This might be a shortage as any resource can evolve and metadata about it will vary along time.

About the 3D object description language VRML/X3D [22, 23], the metadata specification is also taken into account. The WorldInfo node type in the VRML standard

is used to capture general descriptive information about an entire scene. Supporting a finer level, the X3D standard includes the "metadata" field into each of its node types for more description about the node (instance) which is being considered. The value of this field is an instance of one of the node types: MetadataDouble, MetadataFloat, MetadataInteger, MetadataSet, and MetadataString. Such support of the VRML/X3D standard allows the incorporating of the metadata part into the graphical part of 3D objects and their scenes. These two parts are gathered together in the resultant graphical data files. Consequently, a file processing approach would be employed for manipulations on these two parts of 3D objects and their scenes. Every action related to the incorporating of the metadata part into the graphical part of 3D objects and their scenes is left for graphics designers when they construct 3D objects and their scenes. Every action related to the consumption of 3D objects and their scenes is left for front-end application developers at the application level. This incorporation of the metadata part into the graphical data part of 3D objects and their scenes makes the scene graph totally loaded and traversed whenever metadata about 3D objects and their scene is requested. Furthermore, these graphics designers and front-end application developers need to reach a consensus on the representation of metadata about 3D objects and their scenes as well as the content of the resulting graphical data files.

4.5 Summary

This chapter provides the fundamentals of the theory related to our research. Basic notions of a database, a data model, a temporal database, and a spatio-temporal database have been introduced. The advantages of a database system over a file processing system are figured out to give us the rationale behind our focus on an object relational database system for the handling of a large number of 3D objects and their scenes. The database system should be capable of dealing with the temporal and spatial characteristics of 3D objects which are gathered in scenes. From the examination on metadata issues and several metadata standards, both graphical data and metadata parts of 3D objects and their scenes should be taken into consideration with regard to valid time.

Chapter 5

The Handling of 3D Objects

This chapter presents an overview on the handling of 3D objects with consideration on temporal aspects at both graphical data and metadata levels. In this work, we concentrate on 3D visualization features of 3D objects. A *three dimensional (3D) object* is an object of interest in the universe of discourse (UoD) that is graphically represented by means of a geometry and its appearance for visualization or by composing other 3D objects in a 3D space. These 3D objects are gathered to form a 3D scene; thus, spatially related to each other. In addition, we treat an entire 3D scene in a 3D graphical data source which is composed of one or many 3D objects as a composite 3D object. A 3D object with some temporal aspect supported for its properties, relationships, and/or itself is called a *temporal 3D object*. The temporal aspect supported in this work is *valid time* because valid time is the time towards the real world. So, it is relevant for us to consider valid time rather than transaction time. Once a 3D object is temporal, metadata about it is also temporal to reflect temporal features of the object.

5.1 The Proposed Procedure for the Handling of 3D Objects with Valid Time

In this subsection, a procedure for the handling of 3D objects with valid time is proposed to automatically achieve a temporal database from a 3D graphical data source which is true in the modeled world for some period of time. The validity of a source is given as an input together with the 3D graphical data source. The proposed procedure shown in Figure 5.1 is explained as follows.

Regardless of how 3D objects and their scenes are created and formatted, we only pay attention to the visualization features of 3D objects. These objects and their scenes can be obtained from 3D authoring tools, 3D programming toolkits, or traditional 3D object recognition and analysis methods on real scenes. In this procedure, the handling of 3D objects consists of two parts: the handling of 3D graphical data and the handling of metadata. The handling of 3D graphical data starts with a 3D scene graph in terms of graphical elements and the handling of metadata starts with a scene graph in terms of 3D objects each of which is recognized as some graphics encoding fragment in the

input data source after the handling of 3D graphical data. Therefore, the handling of metadata about 3D objects and their scenes in our work is independent of the graphical data format of the 3D graphical data source. It is noted that 3D object recognition is not in the scope of this research. The 3D object recognition topic can be found in [35].

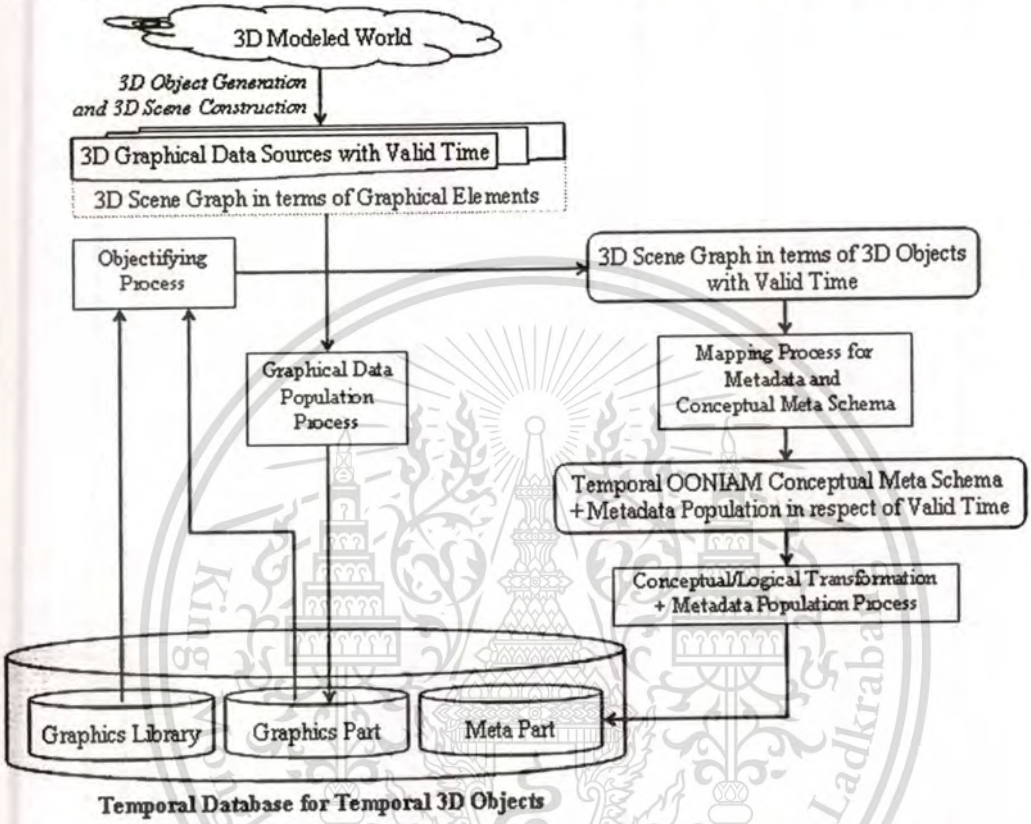


Figure 5.1 The Proposed Procedure for the Handling of 3D Objects with Valid Time

For the handling of the 3D graphical data part of 3D objects and their scenes, the 3D scene graph of the input 3D graphical data source is examined in terms of graphical elements. All graphical information of the input 3D scene graph will be stored in the graphics part of the temporal database for temporal 3D objects by the graphical data population process presented in section 10.1. This graphical data part is then checked with the 3D patterns in the graphics library to identify which graphical data fragment in the input 3D graphical data source represents a meaningful 3D object during the objectifying process. The output of this objectifying process is a 3D scene graph in terms of 3D objects with valid time that is given with the input 3D graphical data source.

For the handling of the metadata part of 3D objects and their scenes with regard to valid time, the 3D scene graph of an input 3D graphical data source is now ready in

terms of 3D objects after the handling of the 3D graphical data part. Each source file is fed to our system with some period of time for the validity of the 3D graphical data source in the modeled world. If not specified, the validity is supposed to be from NOW to the infinity, i.e. 12/31/9999. After the mapping process for metadata and a conceptual meta schema, the outcome is a temporal conceptual meta schema based on our temporal object oriented NIAM conceptual schema model for each scene graph of the source. The structure of metadata is well-defined in respect of valid time at the abstract level. Along with the temporal conceptual meta schema, metadata is also extracted corresponding to 3D objects from the 3D graphical data source. At this stage, metadata is called explicit metadata. Valid time of explicit metadata is the same as the one specified for the source. The representation of other metadata, called non-explicit metadata, is supplemented up to particular application requirements if any. This step of the proposed procedure is presented in chapter 7. In chapter 9, our conceptual/logical transformation procedure is invoked on the temporal conceptual meta schema to systematically and automatically obtain a temporal meta database schema for non-temporal/temporal non-explicit/explicit metadata population. Temporal explicit metadata population is automatically carried out during the process of the input data source in chapter 10 while non-temporal/temporal non-explicit metadata needs to be inserted into the metadata part of the temporal database later.

As soon as a temporal database for temporal 3D objects is available, the querying of temporal 3D objects and their scenes is enabled at both graphical data and semantic data levels in many aspects that will be presented in chapter 11. Front-end applications can conveniently have access to these objects via metadata about them by using the standard object relational SQL language extended for temporal and 3D data supports.

For illustration purpose, a case study will be shown in the next chapter Chapter 6. In addition, chapter 11 will present sample queries on the temporal object relational database obtained for the case study.

In this work, a *temporal database for temporal 3D objects* is now defined as a combination of the graphical data and metadata parts of temporal 3D objects, their scenes in which the objects participate, and their relationships with respect to valid time.

5.2 The 3D Graphical Data Part

The 3D graphical data part of a 3D object includes all graphical elements that form a visual presentation of the 3D object to human-beings. The handling of the 3D graphical data part of 3D objects is fixed for all applications that use the same 3D graphical data format. This is because 3D objects of any application are all described in terms of the same set of known graphical elements. So, a temporal database for temporal 3D objects in an application domain differs from the one in another application domain in metadata about temporal 3D objects specific for each application domain.

As assumed in chapter 1, we employ the Virtual Reality Modeling Language (VRML) [22] for handling the graphical data part of temporal 3D objects. In case the input 3D graphical data source is not encoded in VRML, i.e. 3D objects are formatted in some special format different from VRML, a format converter to VRML is needed. As shown in Figure 5.2, a 3D scene graph of the input 3D graphical data source is now a graph composed of VRML nodes according to their transformation relationships. Each node is self-descriptive, used to describe a 3D object or a property of an object, defined on a VRML node type, including one or many fields each of which is of a VRML field type. A node refers to another one through a field of the SFNode or MFNode field type to form an arc in the scene graph.

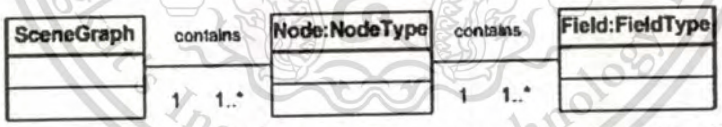


Figure 5.2 The UML Class Diagram describing a Scene Graph at the Graphical Data Level

Technically speaking, a 3D object is represented in VRML by means of a node of the Shape, Transform, Group, LOD, or Switch node type. A node of the Shape, LOD, or Switch node type is non-decomposable while a node of other node types (Transform or Group) is decomposable as a grouping node with a list of children nodes each of which is in turn a node of the Shape, Transform, Group, LOD, or Switch node type that might represent another 3D object. The graphical representation of a 3D object is displayed in Figure 5.3 in terms of graphical elements, i.e. fields and nodes in VRML. If graphically represented by a Shape, LOD, or Switch node, a 3D object is a non-composite object whereas a composite object if by a Transform or Group node.

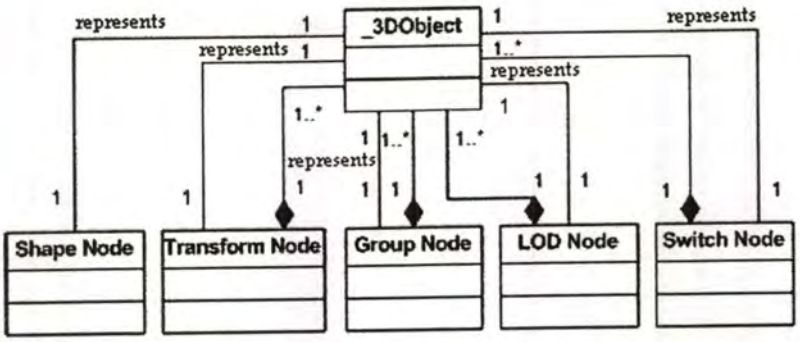


Figure 5.3 The UML Class Diagram describing the Graphical Representation of a 3D Object

With object relational database technology, the handling of the 3D graphical data part of temporal 3D objects is enabled by means of user-defined types, i.e. structured types [17]. The details of the 3D graphical data part of a temporal database for temporal 3D objects will be presented in subsection 9.1.

5.3 The Metadata Part

Metadata about temporal 3D objects is classified into explicit metadata and non-explicit metadata. *Explicit metadata* can be automatically extracted from a 3D graphical data source. In contrast, *non-explicit metadata* cannot be derived from a 3D graphical data source. Instead, non-explicit metadata can be supplied by users, graphics designers, and domain experts.

Explicit metadata includes semantic information about an object type of each 3D object, a category type associated with each object type for categorization of 3D objects of the same object type, associations and compositions among 3D objects, a bounding box of each 3D object for positioning, and spatial relationships among 3D objects. They are all examined with respect to valid time. Figure 5.4 graphically shows a UML class diagram describing temporal explicit metadata about temporal 3D objects. The LS notation is used to denote valid-time life spans [19] of 3D objects and scenes. The VT notation is used to capture the temporal aspect which is valid time [19] of a relationship type. It is realized that our so-called explicit metadata is content-based and can be directly derived from a 3D graphical data source.

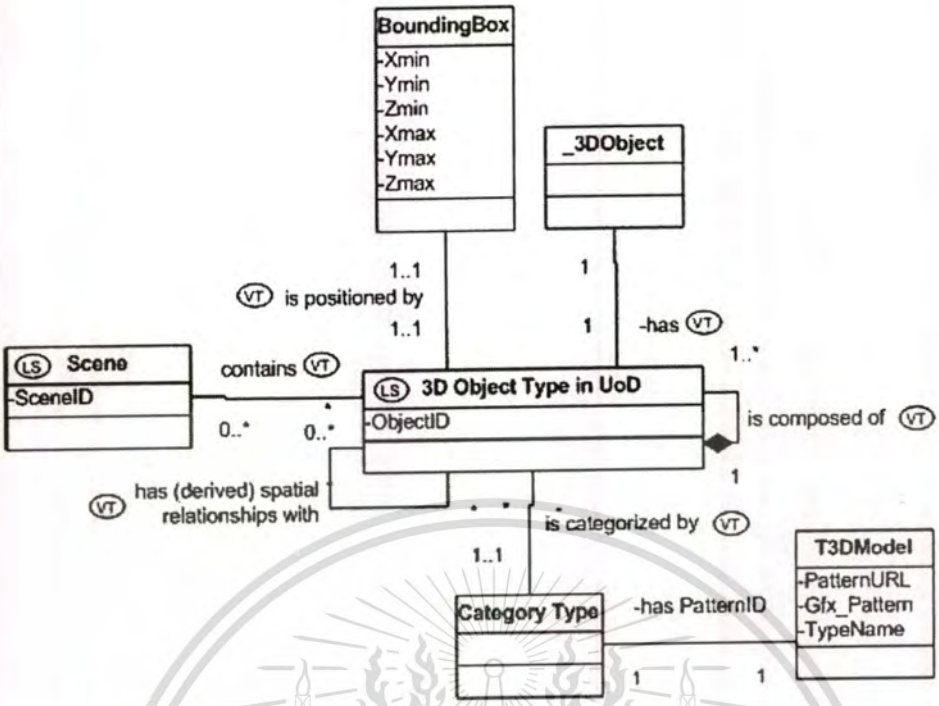


Figure 5.4 The UML Class Diagram describing Explicit Temporal Metadata about Temporal 3D Objects and their Scenes.

Non-explicit metadata consists of the validity of 3D objects given together with their 3D graphical data source and non-presentable descriptive information on 3D objects. The temporal aspect of non-explicit metadata relies on particular application requirements. So, non-explicit metadata can be either non-temporal or temporal.

5.4 Summary

This chapter has figured out the handling of 3D objects where both graphical data and metadata parts of 3D objects are examined with regard to valid time. In the proposed procedure for the handling of temporal 3D objects, the handling for the 3D graphical data part employs the VRML language as an assumption while the handling of the metadata part is independent of the graphical data format of the input 3D graphical data sources. It is noted that 3D object generation, 3D scene construction, and 3D object recognition processes are not part of this work.

Chapter 6

Case Study

For an illustration, we simulate in a 3D space a typical example of land use management adopted and adapted from [29]. The case study uses 3D visualization to display the monitoring about an R region that contains roads, buildings, and areas over the time. It follows the event-driven approach. When a change takes place in the R region, the 3D simulation of the R region will be captured as of that moment and the change will be recorded with the validity from that moment to the infinity (i.e. 12/31/9999) in a corresponding temporal database. The description of the case study is given as follows. Descriptive information about ownerships and other information related to land management are also included. The 3D visualization of the R region is shown in Figures. 6.1..6.3 with three 3D graphical views: perspective, front, and top.

- 01/01/1955: Original information on the R region is held on buildings (b1 and b2) and areas (a1, a2, a3, and a4). Building b1 is located on area a1 and building b2 on area a2. About the ownership, Jeff owns area a1 and building b1, Jane owns area a2 and building b2, John owns area a3, and Julia owns area a4. The 3D simulation of the R region is assumed to be associated with the period of valid time from 01/01/1955 to the infinity for the process on a corresponding 3D graphical data file on 01/01/1955.

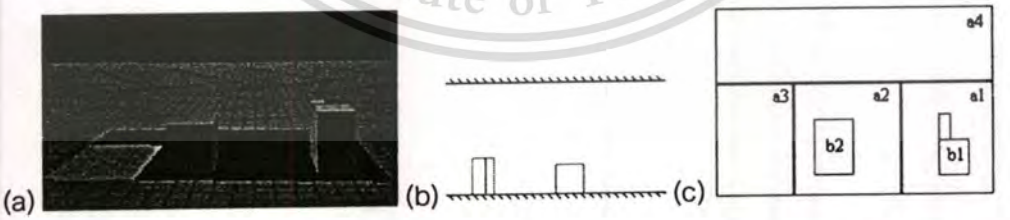


Figure 6.1 3D views of the R region on 01/01/1955: (a) perspective view, (b) front view, (c) top view

- 01/01/1965: Areas a2 and a3 are incorporated to achieve area a5. A new road r1 named Street1 is constructed adjacently to the areas a1 and a5. The owner of area a5 is Jane. The 3D simulation of the R region is assumed to be associated with the

period of valid time from 01/01/1965 to the infinity for the process on a corresponding 3D graphical file on 01/01/1965. The difference between the R region on 01/01/1965 and the R region on 01/01/1955 is required to be realized and valid from 01/01/1965 to the infinity.

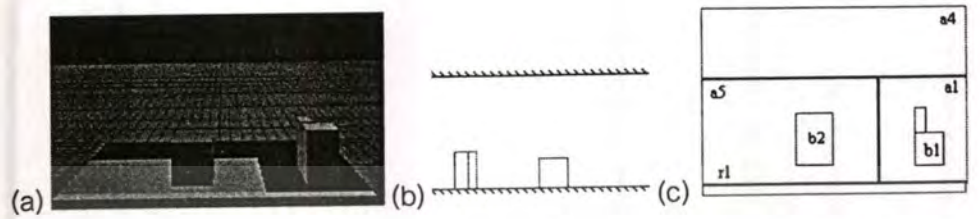


Figure 6.2 3D views of the R region on 01/01/1965: (a) perspective view, (b) front view, (c) top view

- 01/01/1985: Building b2 is demolished. Areas a4 and a5 are enlarged. Building b3 which is a school is built on area a4. The ownership in this R region has no change. The 3D simulation of the R region is assumed to be associated with the period of valid time from 01/01/1965 to the infinity for the process on a corresponding 3D graphical file on 01/01/1965. The difference between the R region on 01/01/1985 and the R region on 01/01/1965 is required to be realized and valid from 01/01/1985 to the infinity.

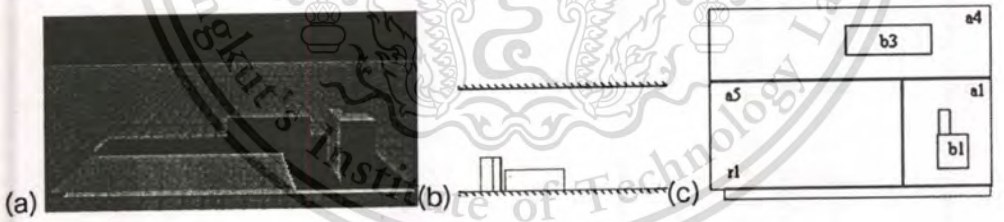


Figure 6.3 3D views of the R region on 01/01/1985: (a) perspective view, (b) front view, (c) top view

As assumed in chapter 1, the graphical data format used in our work is encoded in the VRML language. Figure 6.4 shows the simplified VRML scene graphs of the case study on 01/01/1955, 01/01/1965, and 01/01/1985. The whole scene is described with a node of the Group VRML node type while other graphical objects corresponding to areas (a1, a2, a3, a4, and a5), buildings (b1, b2, and b3), and roads (r1) are described with nodes of the Transform VRML node type. Using a VRML browser, the visualizations of these (a), (b), and (c) scene graphs are displayed in Figures. 6.1a, 6.2a, and 6.3a,

respectively.

```
(a). DEF Region1 Group { # Region Scene on 01/01/1955
        children [
                DEF Parea1 Transform {...}          #a1
                DEF Parea2 Transform {...}          #a2
                DEF Parea3 Transform {...}          #a3
                DEF Parea4 Transform {...}          #a4
                DEF Pbuilding1 Transform {...}      #b1
                DEF Pbuilding2 Transform {...}      #b2
        ]
}

(b). DEF Region1 Group { # Region Scene on 01/01/1965
        children [
                DEF Parea1 Transform {...}          #a1
                DEF Parea4 Transform {...}          #a4
                DEF Pbuilding1 Transform {...}      #b1
                DEF Pbuilding2 Transform {...}      #b2
                DEF Pstreet1 Transform {...}        #r1
                DEF Parea5 Transform {...}          #a5
        ]
}

(c). DEF Region1 Group { # Region Scene on 01/01/1985
        children [
                DEF Parea1 Transform {...}          #a1
                DEF Parea4 Transform {...}          #a4
                DEF Pstreet1 Transform {...}        #r1
                DEF Parea5 Transform {...}          #a5
                DEF Pbuilding3 Transform {...}      #b3
                DEF Pbuilding1 Transform {...}      #b1
        ]
}
```

Figure 6.4 Simplified VRML scene graphs of the case study: (a). 01/01/1955, (b). 01/01/1965, (c). 01/01/1985

It is bewared that the case study in this chapter is rather small and simple. Nevertheless, it is believed that the size of the case study will have no impact on the temporal object relational database system that is going to be proposed for the handling of 3D objects and their scenes with regard to valid time. This is because the functionalities of the proposed system are mainly considered. Also, all sub processes involved in the temporal data population process on 3D graphical data sources with their valid time are intended to be automatically enabled according to the assumptions previously specified in chapter 1. Moreover, performance issues are not included in the scope of this work.

Besides, only simple graphical elements are used in the case study to graphically represent 3D objects including areas, buildings, and roads of a region. The level of

detail of the case study can be considered to be LOD1 [99] with which no any single detail of each 3D object is presented. Instead, 3D block models are used. It is noted that the level of detail plays a very important role in the recognition of individual 3D objects in the scene. However, 3D object recognition is not part of this proposed work. It is supposed that some necessary 3D object recognition process is available so that we can treat an entire fragment of graphical descriptions as a unit graphically representing some 3D object of interest. Therefore, the level of detail does not influence the development of the proposed temporal object relational database system for the handling of 3D objects and their scenes over the time line.

Chapter 7

Conceptual Data Representation

For the conceptual representation of non-temporal/temporal metadata about temporal 3D objects at the abstract level, the temporal object oriented NIAM (TOONIAM) conceptual schema model is proposed in this chapter as a temporal object oriented extension of the NIAM (Nijssen's Information Analysis Methodology) conceptual schema model [77]. Based on this TOONIAM conceptual schema model, a TOONIAM conceptual meta schema is automatically determined from an input 3D scene graph for non-temporal/temporal explicit metadata.

7.1 The Proposed Temporal Object Oriented NIAM Conceptual Schema Model

Defined as a temporal object oriented extension of the NIAM conceptual schema model, the proposed temporal object oriented NIAM (TOONIAM) conceptual schema model allows capturing the temporal aspect of facts in UoD and supporting object orientation of a large number of application domains. It provides richer semantics in data modeling and reduces the details of a conventional NIAM conceptual schema defined for the same purpose with temporal object orientation.

7.1.1 The Object Type Hierarchy in the TOONIAM Conceptual Schema Model

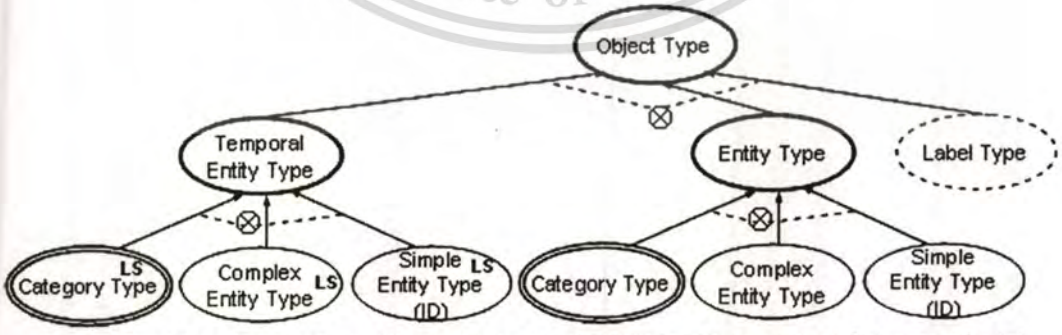


Figure 7.1 The Object Type Hierarchy in the TOONIAM Conceptual Schema Model

The object type hierarchy in the TOONIAM conceptual schema model is described in Figure 7.1. The object oriented extension includes the notions of complex entity types

and category types. The temporal extension introduces the notion of a life span to gain the concept of temporal entity types including temporal simple entity types, temporal complex entity types, and temporal category types. As defined in [19], the life span of an object is the time over which it is defined. For a valid time support, the valid time life span of an object refers to the time when the object exists in the modeled world.

Definition 7.1. A *label type* is a collection of label instances (values or names) used to refer to entities. A label type is graphically denoted as a dashed ellipse.

Definition 7.2. An *entity type* is a collection of objects of interest in UoD. An entity type can be a simple entity type, a complex entity type, or a category type. A *simple entity type* is an original entity type defined in the NIAM conceptual schema model. So, each simple entity type participates in a reference type with a label type whose instances are unique identifiers of the objects of that simple entity type. A *complex entity type* is an entity type with no unique identifier unlike a simple entity type. In addition, a complex entity type may need further definition and have its own methods. A *category type* is a collection of instances that are not individual objects. Instead, they are categories of objects. Further definition and methods might also be required for a category type. Especially, no identifiers are specified for instances of a category type as they are not individual objects in UoD. A simple entity type and a complex entity type are graphically denoted as solid ellipses while a category type as a solid double-line ellipse.

Definition 7.3. A *temporal entity type* is an entity type with its life span graphically denoted with the LS symbol where the entity type itself can be a simple entity type, a complex entity type, or a category type.

Relationship types among simple entity types, complex entity types, and/or category types are also called fact types while relationship types among a simple entity type, a complex entity type, or a category type with some label type are called reference types. Subtype/super type relationships for inheritance among simple entity types are still kept as-is while subtype/super type relationships among complex entity types are also allowed to exist in the TOONIAM conceptual schema model.

7.1.2 The Relationship Type Hierarchy in the TOONIAM Conceptual Schema Model

Along with conventional subtype/super type relationship types, fact types, and reference types, temporal relationship types including temporal fact types and temporal reference types are added into the relationship type hierarchy of the TOONIAM model in Figure 7.2. A temporal relationship type is graphically denoted with the VT symbol.

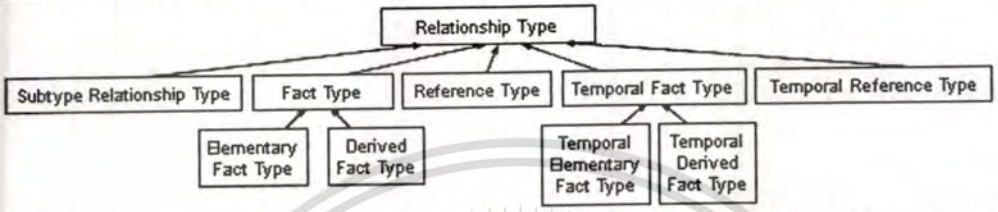


Figure 7.2 The Relationship Type Hierarchy in the TOONIAM Conceptual Schema Model

Definition 7.4. A *temporal fact type* is a fact type whose temporal aspect is considered.

A temporal aspect can be valid time, transaction time or both (bitemporal time). In this temporal extension, only valid time of facts that are true in a modeled world is supported. Thus, a *valid time fact type* is defined as a fact type whose instances are valid (true) over some periods of time and these periods for validity have to be recorded.

Definition 7.5. A *temporal reference type* is a reference type whose instances are valid (true) over some periods of time.

7.1.3 The Notions of a Main Schema and Sub Schemas in the TOONIAM Conceptual Schema Model

In addition to the concept of complex entity types, the notions of main schemas and sub schemas are defined for object orientation.

Each complex entity type, category type, temporal complex entity type, or temporal category type needs to be further defined for their details. Thus, the definition of each complex entity type, category type, temporal complex entity type, or temporal category type forms a *sub schema* corresponding to the defined type. The * symbol is added into the type to denote that the type is currently further considered. All fact types, reference types, temporal fact types, and temporal reference types are private to this type. Methods might also be included in its sub schema and grouped together in a trapezium. This mechanism generates the notion of "object orientation". Indeed, the detail of a

complex entity type, a category type, a temporal complex entity type, or a temporal category type is not shown as encapsulated in its sub schema.

In contrast to sub schemas, there is only one main schema in a TOONIAM conceptual schema. The *main schema* describes relationships among simple entity types, complex entity types, category types, temporal complex entity types, and temporal category types. Label types which are unique identifiers of simple entity types are allowed in the main schema. Other label types and reference types are not allowed to be specified in the main schema. Moreover, subtype/super type hierarchies and nested fact types, i.e. compound object types, are allowed only in the main schema. The main schema brings all sub schemas together with itself to form a complete structure of an application UoD. Therefore, the main schema gives us the overall understanding of an application UoD.

7.1.4 Constraints in the TOONIAM Conceptual Schema Model

Based on the taxonomy of integrity constraints in [74], integrity constraints in the TOONIAM conceptual schema model are classified into inherent, implicit, and explicit constraints. This classification of integrity constraints is summarized in Table 7.1.

Table 7.1 A Classification of Integrity Constraints

Feature	Inherent Constraints	Implicit Constraints	Explicit Constraints
Enforcement level	Structure	Instance (data)	Instance (data)
Purpose for	The correctness of a schema defined on the model	The correctness of an application UoD	The correctness of an application UoD
Specification syntax	No	Yes	Yes
Semantics is defined by	System	System	User
Enforcement conditions are defined by	System	System	User
Specified by	System	User	User
Checking point	Conceptual/Logical Transformation	Change occurrence	Change occurrence

- *Inherent constraints*: Inherent constraints are constraints inherently restricted on the constructs of the model for the correctness of a schema defined on the model. These constraints (restrictions) have been pointed out throughout the specification of our TOONIAM conceptual schema model. Inherent constraints on simple entity types, complex entity types, and category types are that a unique identifier is required for each simple entity type and a unique identifier is required for neither

each complex entity type nor each category type. Other inherent constraints as restrictions on relationship types are determined below in Table 7.2. These inherent constraints are automatically enforced once corresponding constructs are referred to in a TOONIAM conceptual schema and automatically checked once that TOONIAM conceptual schema is transformed to some logical schema at the implementation level.

- *Implicit constraints*: Implicit constraints are constraints implicitly supported (handled) by the system implementing the model and able to be specified by the users in the schema defined on the model according to particular application requirements. Once included in a TOONIAM conceptual schema, these implicit constraints are automatically system-handled and system-checked at the implementation level.
- *Explicit constraints*: Explicit constraints are constraints different from all inherent and implicit constraints defined with the model. The semantics and enforcement of these explicit constraints are normally specified by users according to particular application requirements. They can be stated at the conceptual level along with a corresponding conceptual schema and handled (checked) by CHECK constraints and/or triggers, and/or by application programs at the implementation level.

Table 7.2 Inherent Constraints as Restrictions on Relationship Types

Relationship types	Allowed in the Main Schema	Allowed in Sub Schemas
Reference types	Only for identifiers of simple entity types	Yes
Unary fact types	No	Yes
Binary fact types	Yes	Yes
Nested fact types	Yes	No
N-ary fact types (n>2)	Yes	No
Temporal reference types	No	Yes
Temporal unary fact types	No	Yes
Temporal binary fact types	Yes	Yes
Temporal nested fact types	Yes	No
Temporal n-ary fact types (n>2)	Yes	No
Subtype/super type hierarchies	Yes	No

7.1.4.1 Implicit Constraints

Shown in Figure 7.3, all conventional constraints in the NIAM conceptual schema model are implicit constraints. These constraints are kept as-is and allowed to be specified in a TOONIAM conceptual schema. The conventional NIAM constraints are

mandatory role, uniqueness, domain, occurrence frequency, exclusion, equality, subset, subtype exclusion, and subtype exhaustion.

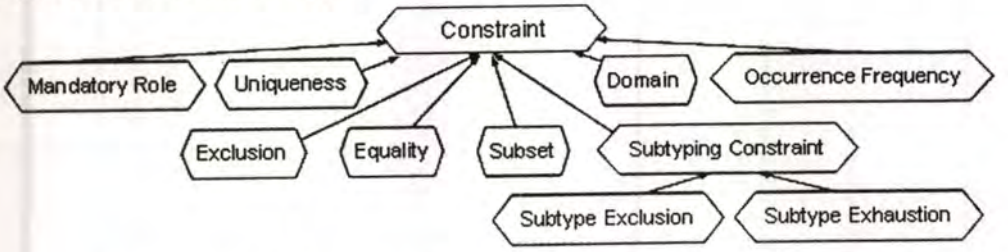


Figure 7.3 Constraint Kinds in the Conventional NIAM Conceptual Schema Model

In this version of the proposed TOONIAM conceptual schema model, we additionally consider other implicit constraints, namely containment constraints and homogeneity constraints, related to the new temporal constructs different from the conventional implicit constraints in Figure 7.3. These implicit constraints can be specified in both main and sub schemas of a TOONIAM conceptual schema. Their semantics and graphical and textual specifications are described as follows.

(a) Containment constraints

This kind of containment constraints is related to the restriction on the life span of an instance of a temporal object type (temporal simple entity type, temporal complex entity type, or temporal category type) with the valid time of its properties and relationships. Consider a temporal object type O1 that participates in a temporal relationship type with the role role1. A containment constraint in Figure 7.4 states that an instance i1 of O1 can participate in that relationship if and only if the instance i1 exists. So, the life span of i1 must contain (include) the time when i1 plays the role role1 in that temporal relationship. Such implicit constraints require no new notation as implicitly system-specified and system-handled for temporal object types and their temporal relationship types.

For each instance i1 of the temporal object type O1,
i1.LS is the life span of i1,
i1.role1.VT is the valid time when i1 plays the role role1,
the condition to enforce this constraint is that i1.LS contains i1.role1.VT.

Figure 7.4 A Containment Constraint

(b) Homogeneity constraints

This kind of homogeneity constraints is related to the homogeneity of the simultaneous participations of an instance of an object type (simple entity type, complex entity type, category type, temporal simple entity type, temporal complex entity type, or temporal category type) in different relationships with different roles. In Figure 7.5, consider the temporal relationship type where the object type O1 plays the role role11 and the temporal relationship type where O1 plays the role role21. A homogeneity constraint states that whenever O1 plays the role role11, O1 also plays the role role21 within the same period of time and vice versa.

For each instance i1 of the object type O1,
i1.role11.VT is the valid time when i1 plays the role role11,
i1.role21.VT is the valid time when i1 plays the role role21,
the condition to enforce this constraint is that i1.role11.VT is equal to i1.role21.VT.

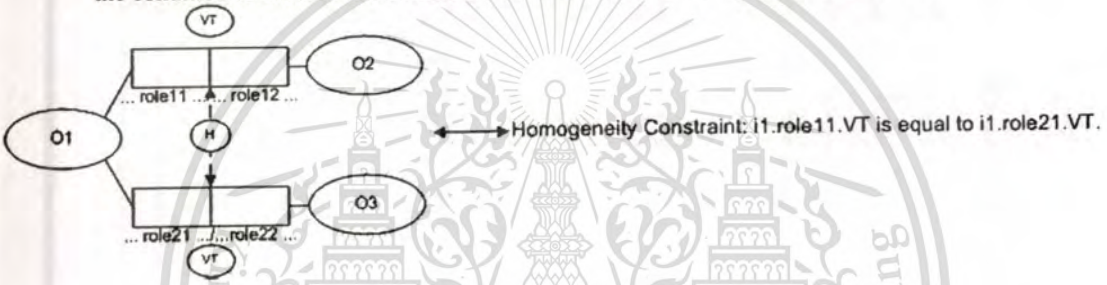


Figure 7.5 A Homogeneity Constraint

This kind of constraints is graphically denoted with the H symbol between two roles of the common object type. In a textual specification, this constraint can be stated below:

HOMOGENEITY FOR EACH <object_type_name>:
<role_name11> (<fact_type_name1>), <role_name21> (<fact_type_name2>).

7.1.4.2 Explicit Constraints

As previously introduced, explicit constraints are constraints explicitly specified by users according to particular application requirements to reflect the correctness of an application UoD. In order to underpin explicit constraints as much as possible at the database level instead of at the application level, we supply in this subsection the specification of a sub set of explicit constraints that are related to our new non-temporal/temporal constructs. Using the introduced specification, an explicit constraint is interpreted as a non-temporal/temporal condition in the form of temporal logic formulae with temporal operators employed from the temporal logic [21]. The explicit constraint can be then transformed along with the associated TOONIAM conceptual

schema during the conceptual/logical transformation procedure. Once specified at the conceptual level in such a way, a sub set of explicit constraints can be enforced by the system at the database level rather than by application programs at the application level. This specification of these explicit constraints can be used in both main and sub schemas of a TOONIAM conceptual schema. Other complex explicit constraints that can not be specified with this explicit constraint specification must be managed by application programs at the application level.

For the formulation of explicit constraints in regard to valid time, we associate the temporal propositional logic with the predicate logic. The predicate logic is employed to refer to objects over which explicit constraints are specified. By means of the predicate logic, each n-ary fact type (n>=1) of n object types is regarded as a predicate with n-arguments each of which takes an instance of a corresponding object type as its value. The temporal propositional logic is used to specify a well-formed formula as a non-temporal/temporal condition that enforces a corresponding explicit constraint. The association is obtained with an AND connective between a well-formed formula of the temporal propositional logic, which expresses an explicit constraint, with other well-formed formulae of the predicate logic. It is bewared that a well-formed formula of the temporal propositional logic is interpreted as a truth value of TRUE or FALSE like any predicate defined in the predicate logic. Thus, the association of the temporal propositional logic and the predicate logic is reasonable.

We first formally define a well-formed formula of the predicate logic. A well-formed formula of the temporal propositional logic is then specified.

(a) The predicate logic

The predicate logic is shown with the alphabet, terms, and well-formed formulae.

- The alphabet includes connectives ($\wedge$ (and), $\vee$ (or), $\neg$ (not)), punctuation symbols ('(', ',', ')'), quantifiers ($\forall$ (universal), $\exists$ (existential)), predicate symbols (p, q, ...), function symbols (f, g, ...), constant symbols, and variable symbols (x, y, ...) where predicate and function symbols can be used in the either prefix or infix fashion, the implication connective ($\rightarrow$) is not included because it can be replaced with the use of the other connectives ($\wedge$, $\vee$, $\neg$).

- The terms are inductively defined as follows.
  - O Each variable symbol is a term.
  - O Each constant symbol is a term.
  - O If f is an n-ary function symbol with n arguments (parameters) t1, t2, .., tn and t1, t2, .., tn are terms, then f(t1, t2, ..., tn) is also a term.
- Atomic formulae of the predicate logic are defined below.
  - O If p is a 0-ary predicate symbol, then p is an atomic formula (a well-formed formula).
  - O If p is an n-ary predicate symbol with n arguments t1, t2, ..., tn for n>0 and t1, t2, ..., tn are terms, then p(t1, t2, ..., tn) is an atomic formula.
- The compound formulae are inductively derived.
  - O If u is an atomic formula, then u is a compound formula.
  - O If u and v are two compound formulae, then so are u∧v, u∨v, ¬u, and ¬v.
  - O If u is a compound formula and x is a variable symbol, then ∀x. u(x) and ∃x. u(x) are also compound formulae. These two compound formulae are intuitively stated as "For all x, u holds" and "There exists an x such that u holds", respectively.

For the semantics with which each n-ary fact type is considered as an n-ary predicate, the interpretation I of each symbol of the alphabet assigns a meaning to the symbol as follows.

- For a constant symbol c, I(c) is an element of some domain Dc.
- For a variable symbol x, I(x) is an element of some domain Dx associated with an object type in a TOONIAM conceptual schema. In other words, I(x) is an instance of an object type.
- For an n-ary function symbol f (t1, t2, ..., tn) for n>=1 and t1, t2, ..., tn are terms, I(f(t1, t2, ..., tn)) = I(f)(I(t1), I(t2), ..., I(tn)) is a function Dt1xDt2x..xDtn → Df. Dt1, Dt2, ..., and Dtn are domains from which I(t1), I(t2), ..., and I(tn) are achieved after the interpretations of the terms t1, t2, ..., and tn. Df is some domain that contains all permitted values after the evaluation of the function. If n is 0, f is a constant symbol.
- For an n-ary predicate symbol p (t1, t2, .., tn) for n>=1 and t1, t2, ..., tn are terms, I(p(t1, t2, ..., tn)) = I(p)(I(t1), I(t2), ..., I(tn)) is a function Dt1xDt2x..xDtn → {TRUE,

FALSE}. If n is 0, then p is a proposition interpreted as either TRUE or FALSE. The predicate symbol p can be the name of some n-ary fact type for n >= 1 in a TOONIAM conceptual schema.

Using the meaning interpretation of the symbols, the interpretation of a well-formed formula of the predicate logic is obtained by assigning a truth value (TRUE or FALSE) to the formula.

- For a proposition (0-ary predicate) p: p=TRUE iff I(p) is TRUE.
- For an n-ary predicate p (t1, t2, ..., tn) for n>=1 and t1, t2, ..., tn are terms: p (t1, t2, ..., tn) = TRUE iff I(p)(I(t1), I(t2), ..., I(tn)) is TRUE.
- For two formulae u and v: u∧v = TRUE iff u = TRUE and v = TRUE.
- For two formulae u and v: u∨v = TRUE iff u = TRUE or v = TRUE.
- For a formulae u: ¬u = TRUE iff u = FALSE.
- For a formulae ∀x. u(x): ∀x. u(x) = TRUE iff for all I(x) of the domain Dx, u(I(x)) = TRUE.
- For a formulae ∃x. u(x): ∃x. u(x) = TRUE iff there exists I(x) of the domain Dx and u(I(x)) = TRUE.

(b) The temporal propositional logic

The presentation of the temporal propositional logic is similar to the one of the predicate logic shown above. The alphabet and well-formed formulae of the temporal propositional logic are also defined in association with a TOONIAM conceptual schema.

- The alphabet consists of connectives (and, or, not), punctuation symbols ('(', ',', ')'), proposition symbols (tp, tq, ...), and temporal operator symbols (G (always), F (eventually), X (next), P (previous), U (until)). Each proposition symbol is a representative of a well-formed formula of the predicate logic presented above where each variable in that predicate logic formula is replaced with its value so that that formula is turned into a proposition. This is obtained from the fact that the proposition is an instantiation of a predicate.
- Atomic formulae of the temporal propositional logic are defined.
  - O Each proposition symbol tp is an atomic formula.

- Gtp, Ftp, Xtp, or Ptp is an atomic formula if tp is an atomic formula where G, F, X, and P are temporal operator symbols introduced previously.
- U(tp, tq) is an atomic formula if tp and tq are two atomic formula where U is a temporal operator symbol introduced above.

- The compound formulae are inductively derived.
  - If u is an atomic formula, then u is a compound formula.
  - If u and v are two compound formulae, then so are $u \wedge v$, $u \vee v$, $\neg u$, and $\neg v$.

For the semantics, we first define the linear time line over the domains D1, D2, ..., and Dk corresponding to all object types in a TOONIAM conceptual schema. The time line is discrete and open towards both the past and the future where the smallest instant is –INFINITY (i.e. 01/01/0001) and the greatest instant is +INFINITY (i.e. 12/31/9999). The special instant NOW separates the time line into the past and the future. The partitioning of the time line is based on some granularity.

Second, we consider the temporal aspect of a proposition symbol tp of the temporal propositional logic specified over a TOONIAM conceptual schema with the involvement of temporal relationship types and their non-temporal/temporal object types. It is noted that each n-ary fact type with the participations of n object types O_1, O_2, ..., and O_n of a TOONIAM conceptual schema is considered as an n-ary predicate P with n arguments corresponding to n object types. That is, each argument of P takes an instance i of the object type (O_1, O_2, ..., or O_n) as its value. If the fact type is temporal, then each instance of that fact type with the participations of n instances i_1, i_2, ..., and i_n of n object types O_1, O_2, ..., and O_n respectively is associated with some periods of time for the validity of that relationship, which is an instance of the fact type, in the modeled world. At that time, the n-ary predicate P becomes $P(i_1, i_2, ..., i_n)$ which is now a proposition with the truth value TRUE and associated with the same periods of time as the instance of the corresponding fact type. Further, each instance i_1, i_2, ..., or i_n is also associated with the same periods of time. Hence, if such instances i_1, i_2, ..., and i_n participate in the formula that is represented by a proposition tp of the temporal propositional logic being considered, the proposition tp is time-dependent, i.e. associated with some periods of time. With this temporal aspect, the proposition tp is assigned a truth value (TRUE or FALSE) at each point in time in common included in the

valid times associated with all instances like i_1, i_2, ..., and i_n involved in the formula represented by the proposition tp. Using the interpretation of a proposition of the temporal propositional logic, the interpretation of a well-formed formula will assign a truth value (TRUE or FALSE) to the formula as follows.

- For a proposition tp: tp = TRUE iff the interpretation of tp is TRUE at the point in time NOW in common, i.e. currently TRUE.
- For an atomic formula Gtp: Gtp = TRUE iff the interpretation of tp is TRUE at every point in time in common, i.e. tp always holds over the time line.
- For an atomic formula Ftp: Ftp = TRUE iff the interpretation of tp is TRUE at least at one point in time in common, i.e. tp holds some time over the time line.
- For an atomic formula Xtp: Xtp = TRUE iff the interpretation of tp is TRUE at the point in time in common that is the next instant of NOW, i.e. tp holds next time over the time line.
- For an atomic formula Ptp: Ptp = TRUE iff the interpretation of tp is TRUE at the point in time in common that is the previous instant of NOW, i.e. tp holds last time over the time line.
- For an atomic formula U(tp, tq): U(tp, tq) = TRUE iff the interpretation of tq is TRUE at some point $point_{tq}$ in time in common and the interpretation of tp is TRUE at every point in time in common till that moment $point_{tq}$, i.e. tp holds till the moment when tq holds over the time line.
- For a formula u∧v where u and v are two formulae: u∧v = TRUE iff u = TRUE and v = TRUE.
- For a formula u∨v where u and v are two formulae: u∨v = TRUE iff u = TRUE or v = TRUE.
- For a formula ¬u where u is a formula: ¬u = TRUE iff u = FALSE.

With the combination of the predicate logic and the temporal propositional logic, an explicit constraint is now specified as a well-formed formula of the predicate logic over a TOONIAM conceptual schema.

Apart from the temporal operators of the temporal propositional logic, Allen's temporal operators [100] are also able to be included into the well-formed formulae of the predicate logic to express an explicit constraint with a direct reference to the valid

time of an instance of each fact type. The Allen's temporal operators are 'before', 'equal', 'meets', 'overlaps', 'during', 'starts', and 'finishes'.

For an illustration on the specification of explicit constraints using the aforementioned logics over a TOONIAM conceptual schema, several sample explicit constraints of the case study will be presented in the following subsection 7.2.3.2.

7.2 The Derivation of a Temporal Object Oriented NIAM Conceptual Meta Schema for 3D Objects with regard to Valid Time at the Abstract Level

With the proposed temporal object oriented NIAM (TOONIAM) conceptual schema model, a TOONIAM conceptual schema that comprises a main schema and several sub schemas is formed as a temporal conceptual meta schema about temporal 3D objects. A temporal conceptual meta schema represents non-temporal/temporal metadata about temporal 3D objects at the abstract level. With the assumptions in chapter 1, it can be systematically derived from an input 3D graphical data source with regard to valid time.

The main schema gives us the world abstraction at the most general level and sub schemas give us the object abstraction at the more detailed level. Such a temporal conceptual meta schema plays a role of a common means for communication among users, graphics designers, domain experts, and front-end application developers. It also provides the overall understanding of temporal 3D objects and their scenes via semantics rather than via graphics encoding and visualization. At this abstract level, naming conventions for consistency can be reached from the agreements among graphics designers, domain experts, and front-end application developers in each application domain. Furthermore, the representation of non-temporal/temporal non-explicit metadata about 3D objects and their scenes can be easily added into the temporal conceptual meta schema for more descriptive information on 3D objects and their scenes in respect of valid time. From the corresponding temporal conceptual meta schema, a temporal meta database schema is systematically and automatically derived.

Given a scene graph of 3D objects true in the modeled world for some period of time, a TOONIAM conceptual meta schema is specified as follows.

7.2.1 The Main Schema

The main schema of a TOONIAM conceptual meta schema is determined to represent an entire 3D scene in which 3D objects take part. From this main schema, we can know what 3D object types appear in a 3D scene, compositions between objects of the same or different types, and which category type is associated with each type of 3D objects. The specification of this main schema from the input 3D scene graph is summarized in Figure 7.6.

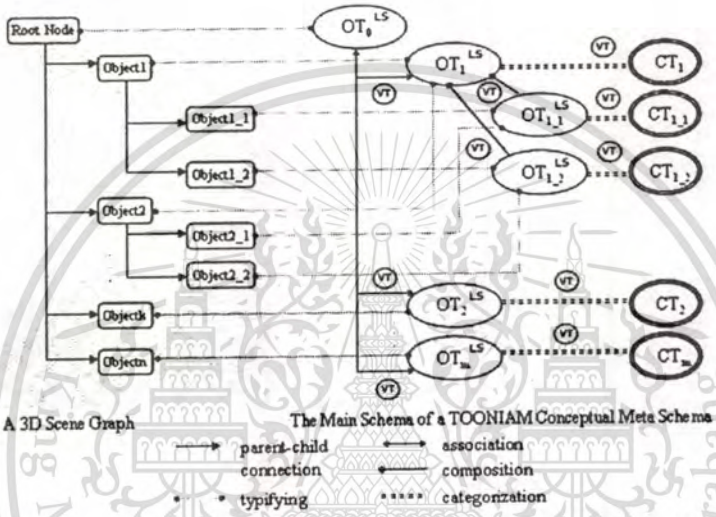


Figure 7.6 The Specification of the Main Schema of a TOONIAM Conceptual Meta Schema from a 3D Scene Graph

The scene graph is now described in terms of 3D objects instead of graphical notions. Relationships between these 3D objects and the root node of the scene graph are made via parent-child connections that form arcs of the scene graph. In this specification, typifying is used to determine a corresponding object type of a 3D object in the scene graph, associations are used to connect an object type corresponding to the entire scene and object types of 3D objects in the scene, compositions are used to specify composite 3D objects, and categorization is used to link an object type of 3D objects and its corresponding category type. They are all considered with valid time.

(1) Object types

Particularly, there is one temporal complex entity type named OT_0 corresponding to an object type representing an entire 3D scene in a 3D data source or an object type representing a collection of 3D scenes of the same structure. For instance, many rooms

in a building with the same decoration are simulated as 3D scenes of the same structure. The OT_0 type is temporal as the valid time for the existence of its entity instances in reality is expected to be captured.

For 3D objects participating in a 3D scene, a group of 3D objects of the same meaningful object type is associated with one temporal complex entity type named OT_i. Similar to the OT_0 type, the OT_i type is temporal as the valid time for the existence of its entity instances in reality is expected to be captured. For example, a 12 storey building in blue and a 9 storey building in white are both associated with one Building temporal complex entity type although their appearances and geometries differ from each other.

In addition, the categorization of 3D objects of the same type is also necessary to enrich data modeling. So, a category type named CT_i is defined for each temporal complex entity type OT_i. Each instance of the category type CT_i plays a role of a category in which 3D objects are of the same appearance and geometry. For example, there are twin towers on some area. The two 3D objects of the Tower temporal complex entity type simulating these twin towers are associated with one instance of the TowerModel category type.

(2) Relationship types and constraints

After the determination of temporal complex entity types (OT_0 and OT_i) and category types (CT_i), we establish relationship types among these types.

Firstly, temporal associations between the scene and its 3D objects are represented by a one-to-many temporal binary fact type. This fact type is established between the temporal complex entity type OT_0 and each temporal complex entity type OT_i. The role of the OT_0 type is on the one side and the role of the OT_i type is on the many side in order to say that for some periods of time, one scene represented by the OT_0 type might contain zero or many 3D objects represented by the OT_i type.

Secondly, if one 3D object is composed of other 3D objects, the temporal composition relationship types among their corresponding temporal complex entity types are defined through one-to-many temporal binary fact types between the temporal complex entity types OT_i and OT_j. The OT_i type corresponding to the class of composite 3D objects is on the one side and the OT_j type corresponding to the class of 3D

object(s) forming a composite 3D object is on the many side. The existence of composite 3D objects is up to how fine 3D graphical objects are to be meaningful. For example, we want a graphical object to be only one semantic object: "table" or to be 5 semantic objects: one "table top" and four "table legs". In the first case, there is only one object type "Table" while in the second case, there are three object types "Table", "TableTop", and "TableLeg" and the composition relationship types exist between the "Table" object type and the "TableTop" object type, between the "Table" object type and the "TableLeg" object type.

Thirdly, the categorization of 3D objects of the same OT_i type is considered. A one-to-many temporal binary fact type between a temporal complex entity type OT_i and a category type CT_i is defined to show a temporal association between temporal 3D objects of the OT_i type and their categories, instances of the category type CT_i. The fact type is temporal as a 3D object is associated with a category for some periods of time. The role of the OT_i type is on the many side while the role of the CT_i type on the one side to say that one or many objects belong to the same category for some periods of time.

In addition to these explicit relationship types, spatial relationships among 3D objects in a scene for some periods of time can be derived by the use of our defined 3D spatial functions in queries over explicit metadata about temporal 3D objects of interest. As compared to the works on metadata about 3D objects in [5, 6, 7, 8, 10] which do not support spatial relationships, in [9, 11] which provide only a few spatial relation kinds, and in [12, 13] which enable the checking of spatial relationships about direction, our work allows a large number of many spatial relationships among 3D objects in a scene to be flexibly checked from the extracted explicit metadata in respect of valid time.

7.2.2 The Sub Schemas

Each temporal complex entity type (OT_0 or OT_i) and each category type CT_i present in the main schema must be further detailed in their own sub schemas.

(1) The sub schema of the temporal complex entity type OT_0

The sub schema of the OT_0 type contains at least one reference type to identify each instance of the OT_0 type that is an entire scene. As shown in Figure 5.3, the label

type involved in that reference type is named SceneID. The value assigned to an instance of this label type can be user-specified or system-generated.

Additional information about other properties of the scene can be captured as non-explicit metadata about the scene and represented in this sub schema. Methods can also be added into this sub schema for operations at the world (scene) level.

(2) The sub schema of a temporal complex entity type OT_i

The definition of the sub schema is private to the OT_i type for the detailed description about 3D objects represented by the OT_i type.

At least one reference type is defined to identify each 3D object in a scene by graphics designers. As shown in Figure 5.3, the label type involved in that reference type is named ObjectID. The value assigned to an instance of this label type can also be user-specified or system-generated.

A one-to-one temporal binary fact type is defined between the OT_i type and another complex entity type _3DObject. This _3DObject type is used to represent a 3D graphical description of an instance (a 3D object) of the OT_i type. The internal definition of the _3DObject type is implementation-dependent up to graphics encoding. In our work, a value assigned to an instance of the _3DObject type is a reference to a graphical description of the corresponding instance (a 3D object) of the OT_i type captured in the graphical data part of a temporal database for temporal 3D objects. This binary fact type is temporal because a 3D object can be of different visual presentations, have various appearances, and located at a few places in a scene along the time. However, at each point in time, the visualization of a 3D object is unique and distinguished from other 3D objects in a scene by geometry, appearance, and/or position.

As for a position of each 3D object in a scene, a one-to-one temporal binary fact type is established between the OT_i type and another complex entity type BoundingBox which is defined to represent minimum bounding boxes of 3D objects. This temporal fact type expresses a temporal association between a 3D object and its position in a 3D scene through its minimum bounding box. In other words, a 3D object might have different positions, i.e. be associated with different bounding boxes, during periods of time when present in a scene. The representation of bounding boxes is then detailed in

another sub schema of the complex entity type BoundingBox. Although the internal definition of the complex entity type BoundingBox is not shown here, a value assigned to an instance of this complex entity type BoundingBox must include the minimum coordinate (Xmin, Ymin, Xmin) and the maximum coordinate (Xmax, Ymax, Zmax) as described in Figure 5.3. These two minimum and maximum coordinates are computed from all coordinates of a 3D object. In the implementation of our work, a DBMS-supplied 3D spatial data type is employed as the data type of bounding boxes.

Other binary relationship types can be determined in this sub schema to represent non-temporal/temporal non-explicit metadata about 3D objects represented by the OT_i type. Methods can be specified in this sub schema for operations at the object level.

(3) The sub schema of a category type CT_i

The sub schema of a category type CT_i provides the details of patterns (categories) represented by the CT_i type. Each instance of the CT_i type is associated with an actual 3D graphical pattern from which a group of 3D objects represented by the temporal complex entity type OT_i corresponding to the CT_i type are created.

A reference type with the label type PatternID is added in this sub schema to uniquely identify each instance of the category type CT_i that is a category. A value assigned to an instance of this label type is a reference to a 3D pattern in the library supplied by graphics designers and/or creators of the library of 3D patterns.

Besides, the sub schema of the CT_i type can also include extra relationship types and methods for non-temporal/temporal non-explicit metadata about patterns of 3D objects represented by the corresponding object type OT_i. These relationship types and methods are supplied up to particular requirements of each application domain.

7.2.3 Constraints

During the mapping of a 3D scene graph into a TOONIAM conceptual meta schema to represent metadata about 3D objects and their scene with valid time at the abstract level, uniqueness constraints are taken into account. They are automatically specified in the resulting TOONIAM conceptual meta schema. In this subsection, more implicit and explicit constraints can be determined from the discussion among users, graphics designers, front-end application developers, and domain experts.

7.2.3.1 Implicit Constraints

Any implicit constraint introduced in subsection 7.1.4.1 can be added into the resulting TOONIAM conceptual meta schema except for containment constraints. Containment constraints will be automatically defined by our system once the TOONIAM conceptual meta schema is processed by the conceptual/logical transformation procedure. An overall examination on possible containment constraints for each temporal object type in the TOONIAM conceptual meta schema are described below.

(a) For the temporal complex entity type OT_0

At the main schema level, a containment constraint is specified for each temporal relationship type in which OT_0 participates with a temporal complex entity type OT_i.

For each instance i1 of OT_0, i1.LS contains i1.contains_OT_i.VT.

At the sub schema level, a containment constraint is specified for each temporal relationship type in which OT_0 participates if any. It depends on each particular application domain.

(b) For each temporal complex entity type OT_i

Similar to the temporal complex entity type OT_0, at the main schema level, a containment constraint is specified for the temporal relationship type in which each temporal complex entity type OT_i takes part with the temporal complex entity type OT_0.

For each instance i2 of OT_i, i2.LS contains i2.is_contained_by_OT_0.VT where 'is_contained_by' is the reversed role of the 'contains' role of the temporal complex entity type OT_0.

At the sub schema level, at least two containment constraints are specified for the temporal relationships with the complex entity types _3DObject and BoundingBox.

For each instance i2 of OT_i, i2.LS contains i2.has_3DObject.VT.
For each instance i2 of OT_i, i2.LS contains i2.has_BoundingBox.VT.

Besides, a containment constraint is specified for each other temporal relationship type in which OT_i participates if any. It depends on each particular application domain.

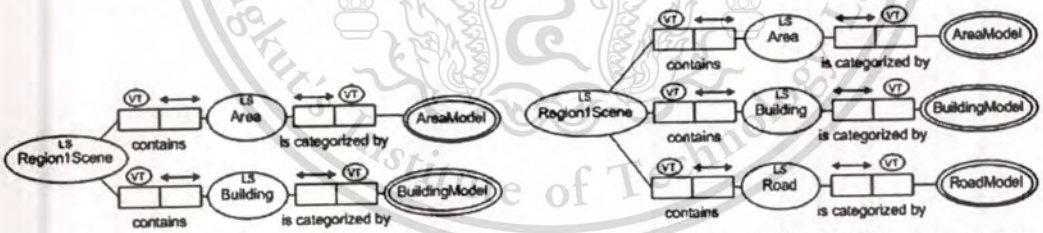
7.2.3.2 Explicit Constraints

By using the specification of explicit constraints in subsection 7.1.4.2, some explicit constraints can be specified over the resulting TOONIAM conceptual meta schema in both main and sub schemas. Their need relies on each particular application domain.

Before going into detail over the case study in chapter 6, we introduce a collection of our 3D spatial functions, which can be utilized to express several explicit constraints related to the positions of 3D objects in a scene, normally called spatial integrity constraints. These 3D spatial functions are based on the standard specification for geographic information in [40]. They are classified for topology, direction, and measurement checking on the bounding boxes of two 3D objects.

- Topological functions: equal3D, disjoint3D, intersects3D, touches3D, crosses3D, within3D, contains3D, overlaps3D, onBoundary3D;
- Directional functions: left3D, on3D, above3D, front3D;
- Measurement functions: distance3Dx, distance3Dy, distance3Dz, distance3D, height3D, width3D, length3D.

Consider the case study along time. A TOONIAM conceptual meta schema is defined on the proposed TOONIAM conceptual schema model in Figures. 7.7..7.10. Relationship types about non-explicit metadata and methods in sub schemas is displayed in italics. The time periods for validity are given for each input data source. The resultant meta schemas are then associated with the same periods of time as the input data source from which they are achieved. If unspecified, the period of time is assumed from NOW to the infinity for validity.



(a) from 01/01/1955 to 01/01/1965 (b) from 01/01/1965 to the infinity

Figure 7.7 The Main Schema of the Case Study

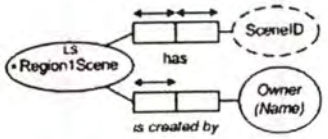
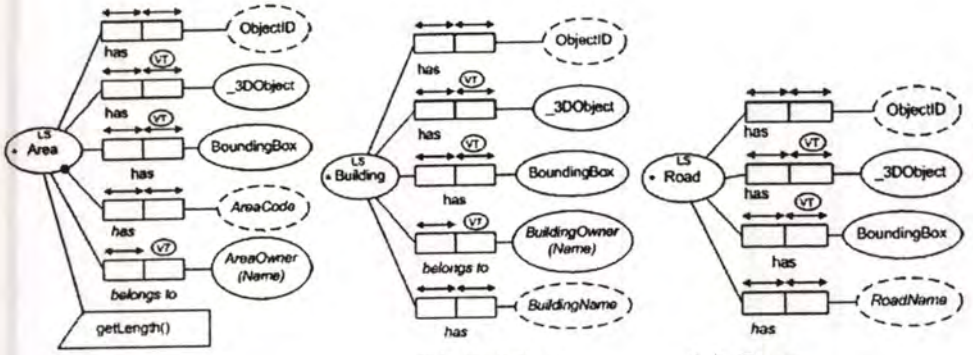
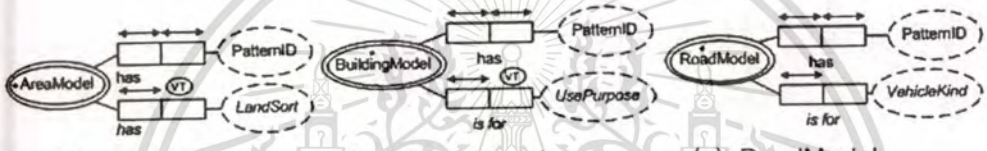


Figure 7.8 The Sub Schema of the Region1Scene Temporal Complex Entity Type from 01/01/1955 to the Infinity



(a). Area (b). Building (c). Road

Figure 7.9 The Sub Schemas of the Temporal Complex Entity Types: (a). Area and (b). Building from 01/01/1955 to the Infinity; (c). Road from 01/01/1965 to the Infinity



(a). AreaModel (b). BuildingModel (c). RoadModel

Figure 7.10 The Sub Schemas of the Category Types: (a). AreaModel and (b). BuildingModel from 01/01/1955 to the Infinity; (c). RoadModel from 01/01/1965 to the Infinity

For integrity constraints over this TOONIAM conceptual meta schema, we define implicit constraints different from those graphically specified in the TOONIAM conceptual meta schema and explicit constraints as follows.

(a) Implicit containment constraints

(a1) In the main schema from 01/01/1955 to 01/01/1965

C1 – For each instance i1 of Region1Scene, i1.LS contains i1.contains_Area.VT.
C2 – For each instance i1 of Region1Scene, i1.LS contains i1.contains_Building.VT.
C3 – For each instance i2 of Area, i2.LS contains i2.belongs_to_Region1Scene.VT.
C4 – For each instance i2 of Area, i2.LS contains i2.is_categorized_by_AreaModel.VT.
C5 – For each instance i3 of Building, i3.LS contains i3.belongs_to_Region1Scene.VT.
C6 – For each instance i3 of Building, i3.LS contains i3.is_categorized_by_BuildingModel.VT.

(a2) In the sub schemas from 01/01/1955 to 01/01/1965

C7 – For each instance i2 of Area, i2.LS contains i2.has_3DObject.VT.
C8 – For each instance i2 of Area, i2.LS contains i2.has_BoundingBox.VT.
C9 – For each instance i2 of Area, i2.LS contains i2.belongs_to_AreaOwner.VT.
C10 – For each instance i3 of Building, i3.LS contains i3.has_3DObject.VT.
C11 – For each instance i3 of Building, i3.LS contains i3.has_BoundingBox.VT.
C12 – For each instance i3 of Building, i3.LS contains i3.belongs_to_BuildingOwner.VT.

(b) Explicit constraints

C13 – Each building must always be constructed totally on one area.

∀i3 of Building, ∀bb3 of BoundingBox, ∃i2 of Area, ∃bb2 of BoundingBox, ∀i22 of Area, ∀bb22 of BoundingBox, i2<>i22 ∧ has_BoundingBox(i3, bb3) ∧ has_BoundingBox(i2, bb2) ∧ has_BoundingBox(i22, bb22) ∧ G(on3D(bb3, bb2)='TRUE') ∧ G(on3D(bb3, bb22)='FALSE').

C14 – The height of each building must be always greater than 10 meters.

∀i3 of Building, ∀bb3 of BoundingBox, has_BoundingBox(i3, bb3) ∧ G(height3D(bb3)>10).

C15 – The school is expected to be built in front of at least one building.

∃i3 of Building, ∃bb3 of BoundingBox, ∀i32 of Building, ∀bb32 of BoundingBox, has_BuildingName(i32, 'School') ∧ i3 <> i32 ∧ has_BoundingBox(i3, bb3) ∧ has_BoundingBox(i32, bb32) ∧ front3D(bb32, bb3)='TRUE'.

C16 – The school must be some time 500 meters near some road.

∃i4 of Road, ∃bb4 of BoundingBox, ∃bb3 of BoundingBox, has_BoundingBox(i4, bb4) ∧ has_BoundingBox('School', bb3) ∧ F(distance3D(bb3, bb4)<500).

C17 – There must be at least one building constructed before the school is built.

∃i3 of Building, ∀i32 of Building, has_BuildingName(i32, 'School') ∧ i3 <> i32 ∧ before(start(lifespan(bb3)), start(lifespan(bb32)))='TRUE'.

7.3 Summary

In this chapter, the conceptual data representation is invented for metadata about 3D objects and their scenes in regard to valid time. New constructs are included into the NIAM conceptual schema model for temporal object orientation that enriches data modeling. Based on the proposed TOONIAM conceptual schema model, a temporal conceptual meta schema is specified to represent non-temporal/temporal metadata about temporal 3D objects and their scenes at the abstract level. This temporal conceptual meta schema allows the overall understanding of 3D objects in a scene at the semantics level instead of at the graphics level. Also, it plays a role of a common means for communication among users, graphics designers, front-end application developers, and domain experts. At this conceptual level, more constraints are able to be stated in the temporal conceptual meta schema. With the support for the specification of non-temporal/temporal explicit constraints, only changes that do not violate any constraint are captured with an appropriate period of time for validity whenever 3D objects change along time.

Chapter 8

The Handling of Temporal Data

8.1 A Sample Database

In order to illustrate the features of our handling of temporal data as generally as possible, the case study about 3D objects is not used in this chapter. Instead, a typical example in [101] is adopted and adapted to suit our valid time support. The Skill temporal attribute of employees is ignored for brevity. As shown below, the time-varying information of employees and the departments where they work is captured. The REFC column is an extra column used for unique identifiers of employees and departments.

In subsection 8.1.1, temporal data is kept with the tuple/object timestamping scheme. The information of a single employee is split into several tuples and into various tables as shown in Tables 8.1..8.4. The information of a single department is similarly split in Tables 8.5..8.7. In contrast, the information of a single employee is naturally gathered into one single tuple in Table 8.8 and the one of a single department in Table 8.9 with the attribute timestamping scheme in subsection 8.1.2. In subsection 8.1.3, both tuple/object and attribute timestamping schemes are employed in Tables 8.10 and 8.11 where the tuple/object timestamping scheme is used to handle the life span of a single entity in the modeled world and the attribute timestamping scheme to handle its time-varying properties and relationships.

8.1.1 Temporal Data with the Tuple/Object Timestamping Scheme

Table 8.1 The Emp1 Temporal Table

REFC	Gender	D_birth
E1	M	7/1/1955
E2	F	10/1/1960
E3	M	1/1/1962
E4	M	5/15/1950
E5	F	12/1/1960

Table 8.2 The Emp2 Temporal Table

REFC	Name	vtStart	vtEnd
E1	Ed	2/1/1982	1/1/1988
E1	Edward	1/1/1988	12/31/9999
E2	Di	1/1/1982	12/31/9999
E3	John	1/1/1962	1/1/1978
E3	Johnson	1/1/1978	12/31/9999
E4	Jack	5/15/1950	12/31/9999
E5	White	12/1/1960	12/31/9999

Table 8.3 The Emp3 Temporal Table

REFC	Salary	vtStart	vtEnd
E1	20	2/1/1982	6/1/1982
E1	30	6/1/1982	2/1/1985
E1	40	2/1/1985	2/1/1987
E1	40	4/1/1987	12/31/9999
E2	30	1/1/1982	8/1/1984
E2	40	8/1/1984	9/1/1986
E2	50	9/1/1986	12/31/9999
E3	45	1/1/1984	1/1/1989
E3	55	1/1/1989	12/31/9999
E4	40	1/1/1980	1/1/1984
E4	50	1/1/1984	12/31/9999
E5	30	1/1/1980	1/1/1984
E5	40	1/1/1984	1/1/1989
E5	45	1/1/1989	12/31/9999

Table 8.4 The Emp4 Temporal Table

REFC	Dept	vtStart	vtEnd
E1	D1	2/1/1982	2/1/1987
E1	D2	4/1/1987	12/31/9999
E2	D1	1/1/1982	12/31/9999
E3	D4	1/1/1984	12/31/9999
E4	D4	1/1/1980	1/1/1984
E4	D3	1/1/1984	12/31/9999
E5	D3	1/1/1980	1/1/1984
E5	D4	1/1/1984	12/31/9999

Table 8.5 The Dept1 Temporal Table

REFC	Department
D1	Toy
D2	Book
D3	Sales
D4	Marketing

Table 8.6 The Dept2 Temporal Table

REFC	Manager	vtStart	vtEnd
D1	E2	1/1/1982	12/31/9999
D2	E1	4/1/1987	12/31/9999
D3	E4	1/1/1984	12/31/9999
D4	E3	1/1/1984	12/31/9999

Table 8.7 The Dept3 Temporal Table

REFC	Budget	vtStart	vtEnd
D1	150	1/1/1982	8/1/1984
D1	200	8/1/1984	1/1/1987
D1	100	1/1/1987	12/31/9999
D2	200	4/1/1987	12/31/9999
D3	150	1/1/1980	1/1/1984
D3	250	1/1/1984	1/1/1987
D3	400	1/1/1987	12/31/9999
D4	120	1/1/1980	1/1/1984
D4	150	1/1/1984	1/1/1989
D4	250	1/1/1989	12/31/9999

8.1.2 Temporal Data with the Attribute Timestamping Scheme

Table 8.8 The Emp Temporal Table with the Attribute Timestamping Scheme

REFC	Name			Gender	D_birth	Salary			Dept		
	vtValue	ValidTime				vtValue	ValidTime		vtValue	ValidTime	
		vtStart	vtEnd				vtStart	vtEnd		vtStart	vtEnd
E1	Ed	2/1/1982	1/1/1988	M	7/1/1955	20	2/1/1982	6/1/1982	D1	2/1/1982	2/1/1987
						30	6/1/1982	2/1/1985			
	Edward	1/1/1988	12/31/9999			40	2/1/1985	2/1/1987	D2	4/1/1987	12/31/9999
							4/1/1987	12/31/9999			
E2	Di	1/1/1982	12/31/9999	F	10/1/1960	30	1/1/1982	8/1/1984	D1	1/1/1982	12/31/9999
						40	8/1/1984	9/1/1986			
						50	9/1/1986	12/31/9999			
E3	John	1/1/1962	1/1/1978	M	1/1/1962	45	1/1/1984	1/1/1989	D4	1/1/1984	12/31/9999
	Johnson	1/1/1978	12/31/9999			55	1/1/1989	12/31/9999			
E4	Jack	5/15/1950	12/31/9999	M	5/15/1950	40	1/1/1980	1/1/1984	D4	1/1/1980	1/1/1984
						50	1/1/1984	12/31/9999	D3	1/1/1984	12/31/9999
E5	White	12/1/1960	12/31/9999	F	12/1/1960	30	1/1/1980	1/1/1984	D3	1/1/1980	1/1/1984
						40	1/1/1984	1/1/1989	D4	1/1/1984	12/31/9999
						45	1/1/1989	12/31/9999			

Table 8.9 The Dept Temporal Table with the Attribute Timestamping Scheme

REFC	Department	Manager			Budget		
		vtValue	ValidTime		vtValue	ValidTime	
			vtStart	vtEnd		vtStart	vtEnd
D1	Toy	E2	1/1/1982	12/31/9999	150	1/1/1982	8/1/1984
					200	8/1/1984	1/1/1987
					100	1/1/1987	12/31/9999
D2	Book	E1	4/1/1987	12/31/9999	200	4/1/1987	12/31/9999
D3	Sales	E4	1/1/1984	12/31/9999	150	1/1/1980	1/1/1984
					250	1/1/1984	1/1/1987
					400	1/1/1987	12/31/9999
D4	Marketing	E3	1/1/1984	12/31/9999	120	1/1/1980	1/1/1984
					150	1/1/1984	1/1/1989
					250	1/1/1989	12/31/9999

8.1.3 Temporal Data with both Tuple/Object and Attribute Timestamping Schemes

Table 8.10 The Emp Temporal Table with both Tuple/Object and Attribute Timestamping Scheme

REFC	Name			Gender	D_birth	Salary			Dept			LIFESPAN	
	vtValue	ValidTime				vtValue	ValidTime		vtValue	ValidTime		vtStart	vtEnd
		vtStart	vtEnd				vtStart	vtEnd		vtStart	vtEnd		
E1	Ed	2/1/1982	1/1/1988	M	7/1/1955	20	2/1/1982	6/1/1982	D1	2/1/1982	2/1/1987	7/1/1955	12/31/9999
						30	6/1/1982	2/1/1985					
	Edward	1/1/1988	12/31/9999			40	2/1/1985	2/1/1987	D2	4/1/1987	12/31/9999		

							4/1/1987	12/31/9999					
E2	Di	1/1/1982	12/31/9999	F	10/1/1960	30	1/1/1982	8/1/1984	D1	1/1/1982	12/31/9999	10/1/1960	12/31/9999
						40	8/1/1984	9/1/1986					
						50	9/1/1986	12/31/9999					
E3	John	1/1/1962	1/1/1978	M	1/1/1962	45	1/1/1984	1/1/1989	D4	1/1/1984	12/31/9999	1/1/1962	12/31/9999
	Johnson	1/1/1978	12/31/9999			55	1/1/1989	12/31/9999					
E4	Jack	5/15/1950	12/31/9999	M	5/15/1950	40	1/1/1980	1/1/1984	D4	1/1/1980	1/1/1984	5/15/1950	12/31/9999
						50	1/1/1984	12/31/9999	D3	1/1/1984	12/31/9999		
E5	White	12/1/1960	12/31/9999	F	12/1/1960	30	1/1/1980	1/1/1984	D3	1/1/1980	1/1/1984	12/1/1960	12/31/9999
						40	1/1/1984	1/1/1989	D4	1/1/1984	12/31/9999		
						45	1/1/1989	12/31/9999					

Table 8.11 The Dept Temporal Table with both Tuple/Object and Attribute Timestamping Schemes

REFC	Department	Manager			Budget			LIFESPAN	
		vtValue	ValidTime		vtValue	ValidTime		vtStart	vtEnd
			vtStart	vtEnd		vtStart	vtEnd		
D1	Toy	E2	1/1/1982	12/31/9999	150	1/1/1982	8/1/1984	1/1/1982	12/31/9999
					200	8/1/1984	1/1/1987		
					100	1/1/1987	12/31/9999		
D2	Book	E1	4/1/1987	12/31/9999	200	4/1/1987	12/31/9999	4/1/1987	12/31/9999
D3	Sales	E4	1/1/1984	12/31/9999	150	1/1/1980	1/1/1984	1/1/1980	12/31/9999
					250	1/1/1984	1/1/1987		
					400	1/1/1987	12/31/9999		
D4	Marketing	E3	1/1/1984	12/31/9999	120	1/1/1980	1/1/1984	1/1/1980	12/31/9999
					150	1/1/1984	1/1/1989		
					250	1/1/1989	12/31/9999		

8.2 Temporal Data Representation

The temporal data management in this research work will be based on the object relational data model. The temporal database language proposed in this chapter will be based on the object relational SQL language [17]. The representation of temporal data will be defined with both tuple/object and attribute timestamping schemes. Nevertheless, temporal data manipulations are mainly demonstrated with the attribute timestamping scheme.

Definition 8.1 (*Temporal object relational database, TDB*). A *temporal OR database TDB* is an OR database *DB* that supports some aspect of time, not counting user-defined time [19]. In terms of tables, a *temporal OR database TDB* is a collection of 1NF, non-1NF, and/or typed tables. At least one of these tables has some temporal aspect. Such a table is called a *temporal table*. If a temporal table is a typed table defined on a structured type, such a structured type is called a *temporal structured type*.

In this chapter, the temporal aspect supported is valid time. By using a temporal OR database *TDB*, elementary facts which are non-decomposed are time-varying and their histories are recorded. With data timestamping at the attribute level, these histories are grouped together for a single entity and the concept of a *temporal attribute* arises. If an elementary fact represented as an entire row/tuple is temporal, the timestamping of

that fact must be placed at the tuple/object level. This case requires the tuple/object timestamping scheme. In our work, it is handled with the support of the valid time life span of an entity where a valid time life span of an entity defined in [19] is the time over which the entity exists in the modeled reality. Besides, it is worth distinguishing the notion of a temporal attribute used in the attribute timestamping scheme from the notion of a temporal attribute used in the tuple timestamping scheme. A temporal attribute in the tuple timestamping scheme is a timestamping attribute such as FROM_DATE and TO_DATE. Differently, a temporal attribute in the attribute timestamping scheme is a timestamped attribute stemming from the time-varying nature of an elementary fact type. Therefore, our temporal data representation mainly aims at a strong notion of the "history [19] of an attribute" of an object/entity of interest as highlighted in [102] so that a temporal table is now a table that contains at least one *temporal column*. In case of a temporal typed table defined on a temporal structured type, the temporal structured type is a structured type that comprises one or many temporal attributes each of which corresponds to a temporal column. For the life span support, if each single row of a table is associated with an entity whose life span needs to be captured, an extra column LIFESPAN is included into that table. If the table is defined on a structured type, the definition of that structured type is appended with an extra attribute LIFESPAN.

For an illustration, the two temporal tables Emp and Dept are shown in Tables 8.8 and 8.9 in subsection 8.1.2, respectively. In the Emp temporal table, Name, Salary, and Dept are temporal columns while Gender and D_birth non-temporal ones. In the Dept temporal table, Manager and Budget are temporal columns while Department is a non-temporal one. These two temporal tables can be temporal non-1NF tables or temporal typed tables defined on temporal structured types Emp_t and Dept_t, respectively. There are temporal attributes Name, Salary, and Dept in the Emp_t temporal structured type while Manager and Budget in the Dept_t temporal structured type. For the life span support, Tables 8.10 and 8.11 show the inclusion of the extra column LIFESPAN.

8.2.1 Temporal Column

Firstly, the *data representation of a temporal column* A_j is defined. It is generalized regardless of an actual data type AT_j of A_j. This general temporal support leads to no

difference between a temporal column of type VARCHAR, a temporal column of type NUMERIC, and a temporal column of type REF, etc. in regard to time-varying.

Consider a data item av_j at the A_j column of a row (av_1, ..., av_j, ..., av_n) in the ut_k table of *DB*. If A_j is not temporal, av_j will be of the AT_j type. If A_j is temporal, the representation of the av_j value must be able to express the time-varying aspect of A_j over the time. At some moment, the value assigned to A_j is true in reality. This observation is based on some partitioning of the time axis known as a time granularity [19]. All observations for A_j form the history of A_j possibly stretching the past, now, and the future. Such a history is called an attribute history [19] of an entity in the modeled world. An attribute history is in fact the av_j value. One observation results in a temporal composition c where $c = (av_{j,c}, vt_{j,c})$:

- c represents the fact that $av_{j,c}$ is true in $vt_{j,c}$ based on some g_j granularity handled as metadata of A_j.
- $av_{j,c}$ is a (snapshot) value of the AT_j data type. $av_{j,c}$ is called the time-dependent component of c.
- $vt_{j,c}$ is a set of disjoint time periods each of which is a closed-open period represented by a starting point and an end point that is not included in the period. $vt_{j,c}$ is called the time component of c.

Definition 8.2 (*Attribute history, av_j*). An attribute history of A_j is a finite collection of temporal compositions: $av_j = \{(av_{j,1}, vt_{j,1}), (av_{j,2}, vt_{j,2}), ..., (av_{j,c}, vt_{j,c}), ..., (av_{j,h}, vt_{j,h})\}$.

Next, *inherent integrity constraints on temporal columns* are specified to guarantee logical data consistency, no value-equivalent duplicate [52], and no data redundancy. First, each period of time contained in any $vt_{j,c}$ satisfies the condition that the starting point in time is less than the end point in time. Second, all values $av_{j,1}$, ..., $av_{j,c}$, ..., and $av_{j,h}$ are of the same type AT_j. Their observations are based on the same granularity g_j. Third, these values $av_{j,1}$, ..., $av_{j,c}$, ..., and $av_{j,h}$ form a set. They are different from each other to avoid value-equivalent duplicates. Finally, the disjointness of $vt_{j,1}$, ..., $vt_{j,c}$, ..., $vt_{j,h}$ is required to prevent two different values $av_{j,c1}$ and $av_{j,c2}$ from being true at the same time for no data redundancy.

Also, we define some *basic functions on a collection of temporal compositions av_j*.

- To retrieve all periods of validity: LIFESPAN(av_j) = UNION($vt_{j,1}$, ..., $vt_{j,c}$, ..., $vt_{j,h}$) where UNION (∪), INTERSECTION (∩), and DIFFERENCE (−) are set operators on sets of periods of time. A result of these operators is also a set of disjoint periods of time. The formal semantics of these operators are defined in [63].
- To retrieve a value true at some moment t in reality: TVALUE(av_j|t) = $av_{j,c}$ where t is included in $vt_{j,c}$
- To access the time-dependent and time components of a temporal composition, i.e. c = ($av_{j,c}$, $vt_{j,c}$), VTVALUE(c) = $av_{j,c}$ and VALIDTIME(c) = $vt_{j,c}$.

If the data representation of a temporal column is based on the relational data model to support the notion of a history, a so-called surrogate key or time-invariant key is required for history identification [19]. The splitting of the data of one single entity might take place. This case might not be expected by database users if such representation is explicit to them or might create some burden to make another natural representation for database users on top of the representation of split relations. However, the difficulty faced in the relational model can be overcome in the nested relational, object oriented, or OR data models which allow the non-atomicity of attributes to represent their histories.

For an illustration, the Salary temporal column of the Emp temporal table in Table 8.8 is examined. Consider the row identified by 'E1'. A corresponding value av_j at the Salary temporal column that is an attribute history is av_j = {(20, {[2/1/1982, 6/1/1982)}), (30, {[6/1/1982, 2/1/1985)}), (40, {[2/1/1985, 2/1/1987), [4/1/1987, 12/31/9999)})} where (20, {[2/1/1982, 6/1/1982)}), (30, {[6/1/1982, 2/1/1985)}), and (40, {[2/1/1985, 2/1/1987), [4/1/1987, 12/31/9999)}) are temporal compositions. In the (20, {[2/1/1982, 6/1/1982)}) temporal composition, 20 is the time-dependent component and {[2/1/1982, 6/1/1982)} is the time component. It is also realized that the value av_j satisfies all the inherent integrity constraints stated above.

8.2.2 Temporal Attribute and Temporal Structured Type

Consider a structured type ST_i for i in 1..m. For j in 1..n, an attribute A_j of ST_i is a temporal attribute if the temporal aspect of A_j is taken into account. As a result, ST_i is a

temporal structured type. Each value assigned to the A_j temporal attribute must obey the temporal data representation of a temporal column that has already been defined.

Apart from so-called temporal attributes, a structured type might contain an extra attribute LIFESPAN so that the life span of each instance (object) of the structured type, i.e. a tuple/row of a typed table defined on the structured type, can be captured. The life span of an instance of a structured type is a set of disjoint periods of time over which the existence of the instance is true in the modeled world. Therefore, the domain of this LIFESPAN attribute is the same as the domain of the time component of a temporal composition in subsection 8.2.1.

8.2.3 Temporal Table

With the notions of a temporal column, a temporal attribute, an extra attribute LIFESPAN, and a temporal structured type, the temporal data representation of a temporal table ut_k of *TDB* is further defined. From the relational database point of view, a temporal table ut_k of *TDB* contains at least one temporal column. If the life span support is required, ut_k will contain an extra column LIFESPAN. From the object database point of view, a temporal table ut_k of *TDB* is defined on a temporal structured type ST_i. All non-temporal and temporal attributes are bundled together. Some attribute histories might depend on each other according to particular application requirements. As a result, columns of ut_k corresponding to the temporal attributes A_j are temporal and others are non-temporal. If ST_i contains the LIFESPAN attribute, ut_k will contain a corresponding column LIFESPAN for the life span support. Each value at the LIFESPAN column of ut_k if required is a set of disjoint periods of time. Each row of ut_k is regarded as a non-encapsulated object of the type ST_i with a REFC value. Each REFC value is naturally used for attribute history identification [66].

Consider operations on temporal tables. In comparison with a conventional table defined in an OR SQL language (SQL:3, SQL:99, or SQL:2003), a temporal table and a non-temporal table only differ from each other in the temporal aspect of their columns. That is, all operations on a temporal table are exactly the same as those on a non-temporal table if the operations are not related to any temporal column or the temporal aspect of involved temporal columns is ignored. Hence, for each operation (query,

insert, delete, or update) on temporal tables, we consider the temporal aspect of one or many columns involved in the operation. It is bewared that the temporal aspect of a column will need to be considered in derived columns (expressions in general) which are returned to users by means of projection. The temporal aspect of a column also needs to be considered in search conditions for selection, join, deletions, and updates. Moreover, each temporal data modification requires some periods of time for applicability provided by users. This requirement is accomplished by extending an appropriate operation (insert, delete, or update) with the specification of valid time. Besides, the existing deletion operation on non-temporal tables performs only at the tuple/object level while specific facts associated with some temporal attributes might be removed on demand. So, a new temporal deletion operation at the attribute level is defined.

Unlike operations on temporal tables which can be derived from their counterparts on non-temporal tables, operations on temporal columns of temporal tables need to be particularly defined because operations on data of different data types are not the same. Thus, we have defined a set of temporal methods associated with temporal data types which are used to form the structure of temporal attributes of any data type. Our temporal methods are then used for the transformation of temporal SQL statements into non-temporal SQL statements executable to an employed ORDBMS. Furthermore, these methods can be directly and non-restrictively used by database users who have deep knowledge of the representation of temporal columns and abilities in handling temporal data all on their own. Above all, the use of our temporal methods can preserve the notion of history orientation which is too hard to be achieved by using the relational and nested relational data models.

8.3 The Proposed Temporal Object Relational SQL Language

8.3.1 An Overview on the Proposed Temporal Object Relational SQL Language

In this section, a temporal object relational SQL language is proposed with a main focus on the attribute timstamping scheme. However, manipulations at the tuple/object level related to the LIFESPAN column of temporal tables can be accomplished in a non-

sequenced manner. The proposed language is a superset of the OR SQL language [17], handling valid time at the attribute level. Particularly, it is a nonprocedural means for temporal database schema definitions, temporal database schema evolution definitions, definitions of integrity constraints on temporal tables and temporal columns, temporal data querying, and temporal data modifications on a temporal OR database which has been defined in section 8.2. With the temporal features provided at the attribute level, the language is comparable to ATSQL [45], which is based on the relational data model with the temporal support at the tuple level. This is because the construction of our language also uses semantic defaults and the denotational-semantics-style mapping of temporal SQL statements to SQL statements enhanced by our defined temporal data types and temporal methods. Nevertheless, the proposed language is completely different from ATSQL and other existing languages following the attribute timestamping scheme as the new constructs of our language for temporal data querying are obtained from temporal logic.

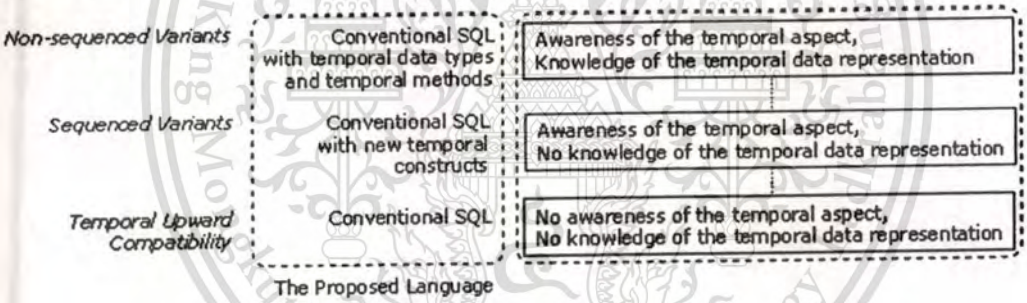


Figure 8.1 The Proposed Temporal Object Relational SQL Language

Shown in Figure 8.1, our language basically satisfies the five requirements in [52].

i. The proposed language provides a valid time support at the attribute level. Even though not included in this language, the semantics of transaction time can later be included as a new extension or using a built-in transaction time support of an existing ORDBMS. Furthermore, valid time and transaction time are orthogonal to each other. The exclusion of transaction time has no impact on our work.

ii. Upward compatibility is ensured. Since the language is developed as a temporal superset of the OR SQL language [17], all existing constructs applied to non-temporal data can operate exactly as before.

iii. Temporal upward compatibility is ensured. The proposed language guarantees that all manipulations on a temporal database expressed with the constructs of an object relational SQL language are evaluated with only the current state of the temporal database. In addition, our language ensures existing non-temporal applications not to be broken if the temporal aspect is required. Temporal database schema evolution definitions are also included in our language. As shown in Figure 8.1, this feature aims at database users who are not aware of the temporal aspect of an underlying temporal database and have no knowledge of the temporal data representation. These users are non-temporal database users. They use existing constructs of the conventional OR SQL language only to manipulate (query, insert, delete, and update) the underlying temporal database. At that moment, the temporal database is treated as a non-temporal database.

iv. Sequenced variants for main constructs of a database language including queries, modifications (insertions, deletions, and updates), and constraints through the history oriented notion are provided. The concept "sequenced" is adopted in our language for temporal data manipulations on temporal attributes. Figure 8.1 shows that database users are aware of the temporal aspect of a temporal database and capable of using these sequenced variants to facilitate their temporal data manipulations on temporal databases. They are temporal database users.

v. Non-sequenced variants for the notion of non-restrictiveness [45] are supported in the proposed language through the direct use of our temporal data types and temporal methods. In Figure 8.1, a non-sequenced statement of the proposed language is a statement without any sequenced variant. Instead, a non-sequenced statement is simply a conventional OR SQL statement containing our temporal data types and/or temporal methods. With the aid of object relational technology, our temporal data types and temporal methods can be non-restrictively used as built-in and DBMS-supplied types and methods.

As described in Figure 8.1, the language serves a variety of database users with different awarenesses of the temporal aspect of a database and various abilities of data manipulations on a temporal database. Its database users can be both non-temporal and temporal. It provides database users with the strong support for the history oriented

notion when the timestamping is shifted to the attribute level. We mainly focus on the temporal extension for the attribute timestamping scheme while the one for the tuple/object timestamping scheme with the inclusion of the LIFESPAN attribute/column is considered in temporal data definitions and modifications. The syntax of our language is described in the Backus-Naur form (BNF). Only new or modified production rules which are not part of the standard SQL syntax are presented for the formulation of temporal SQL statements with our sequenced variants. Non-terminal symbols are in the form of <xxx> and terminal symbols in the form of XXX.

8.3.2 Temporal Data Definition

8.3.2.1 Temporal Database Schema Definitions

The main focus of the proposed temporal database schema definition is the definition of a temporal structured type and a temporal table which is either a temporal 1NF table, a temporal non-1NF table, or a temporal typed table defined on a temporal structured type. It is known from section 8.2 that the modification of a 1NF/non-1NF table into a temporal one is the inclusion of temporal columns into the existing definition of a 1NF/non-1NF table and the modification of a structured type into a temporal one is the inclusion of temporal attributes into the existing definition of a structured type. Therefore, we modify the production rules <column_definition> and <attribute_definition> in [17] with our new temporal construct VALIDTIME for the specification of the temporal aspect of a column of a 1NF/non-1NF table and the temporal aspect of an attribute of a structured type. Additional information also required in the specification is a granularity (DAY, MONTH, YEAR, etc.) for the observation of the changes of data over time and an optional default period of time for validity. Granularity issues are not minutely examined in this work. Nevertheless, the DBMS-supplied methods related to date and time might be helpful for granularity-based data retrieval.

<column_definition> ::= <non_temporal_column_definition> | <temporal_column_definition>
<temporal_column_definition> ::= <column_name> [<data_type_or_domain_name>]
[<default_clause> | <identity_column_specification> | <generation_clause>]
[<column_constraint_definition>...] [<collate_clause>]
<temporal_aspect>

<attribute_definition> ::= <non_temporal_attribute_definition> | <temporal_attribute_definition>
<temporal_attribute_definition> ::= <attribute_name> <data_type> [<attribute_default>] [<collate_clause>]
<temporal_aspect>

<temporal_aspect> ::= VALIDTIME [<left_paren> <granularity> <right_paren>] [<periods_default>]
<granularity> ::= YEAR | MONTH | WEEK | DAY | HOUR | MINUTE | SECOND | MILISECOND

<periods_default> ::= <periods>
<periods> ::= <a_period> | PERIODS <left_paren> <period_list> <right_paren>
<period_list> ::= <a_period> | <a_period> <comma> <period_list>
<a_period> ::= PERIOD <left_paren> <datetime_expression> <comma> <datetime_expression> <right_paren>

Both <column_definition> and <attribute_definition> can derive either non-temporal or temporal variants. The <non_temporal_column_definition> and <non_temporal_attribute_definition> non-temporal variants are in fact the conventional definitions of a column and an attribute in the non-temporal OR SQL language, respectively. The <temporal_column_definition> and <temporal_attribute_definition> temporal variants are achieved with the addition of <temporal_aspect> at the end of a corresponding definition.

With the new specification of a column, the existing definition CREATE TABLE of a 1NF/non-1NF table is reused for the definition of a temporal 1NF/non-1NF table. With the new specification of an attribute, the existing definition CREATE TYPE of a non-temporal structured type is kept for the definition of a temporal structured type. The definition of a temporal typed table on a temporal structured type is the existing definition CREATE TABLE ... OF ... of a non-temporal typed table.

Regarding semantics, the placement of VALIDTIME in <temporal_aspect> is restricted to only columns in the definition of a temporal 1NF/non-1NF table and attributes in the definition of a temporal structured type that stem from temporal elementary fact types. This restriction is essential to guarantee that a nonprocedural declaration of the temporal aspect of a column or an attribute is meaningful. In particular, each row/tuple of a table comprises one or many elementary facts. If some of these elementary facts grouped together in a row/tuple are time-varying, the timestamping of those facts is placed at the attribute level. Therefore, only columns of a table associated with those elementary fact types can be considered to be temporal with the VALIDTIME clause. If an elementary fact represented as an entire row/tuple is temporal, the timestamping of that fact must be placed at the tuple/object level. This case must be examined with the tuple-timestamping scheme.

For the inheritance among structured types, the OR SQL language allows super type/sub type relationships and inheritance. As a temporal structured type is also a structured type, the inheritance concept is able to be applied to temporal structured types. For a super type/sub type relationship, all temporal attributes of a super type will

be inherited by a sub type by obeying the SQL inheritance rules. So, all sub types of a temporal super type are temporal.

Besides, the difference between the definition of a non-temporal table and a temporal table is the inclusion of the specification of integrity constraints on temporal columns which is not supported by the non-temporal OR SQL language. The declaration of integrity constraints on non-temporal columns is kept unchanged. Hence, our integrity constraint specification on temporal columns can be defined together with the conventional constraint specification of a typed table. This part will be presented in subsection 8.3.2.3.

Example 8.1 - As an illustration on the sample database in subsection 8.1.2, two temporal structured types Emp_t and Dept_t and two temporal typed tables Emp and Dept based on these two types, respectively are defined.

```
CREATE TYPE Emp_t AS (
        Name            VARCHAR (64) VALIDTIME (DAY),
        Gender          CHAR(1),
        D_birth         DATE,
        Salary          NUMERIC VALIDTIME (MONTH));
CREATE TYPE Dept_t AS (
        Department      VARCHAR (64),
        Manager         REF(Emp_t) VALIDTIME (MONTH),
        Budget          NUMERIC VALIDTIME (YEAR));
ALTER TYPE Emp_t
ADD ATTRIBUTE Dept REF(Dept_t) VALIDTIME (MONTH);
CREATE TABLE Dept OF Dept_t (
        REF IS REFC SYSTEM GENERATED,
        CONSTRAINT Dept_un UNIQUE (Department));
CREATE TABLE Emp OF Emp_t (
        REF IS REFC SYSTEM GENERATED,
        CONSTRAINT Name_un VALIDTIME UNIQUE(Name));
```

For the life span support, the definitions of a structured type and a 1NF/non-1NF table are simply expanded with the LIFESPAN clause. Once a table is defined on a structured type specified with the LIFESPAN clause and containing one or many temporal attributes or a 1NF/non-1NF table is defined with the LIFESPAN clause and contains one or many temporal columns, a containment constraint is implicitly specified between the LIFESPAN column and each of the other temporal columns in the table. Containment constraints on a temporal table require that for each tuple in such a table, all valid times at the temporal columns must be contained (included) in the periods of time at the LIFESPAN column of the table. These implicit containment constraints will be automatically checked for each modification operation on a table that contains both LIFESPAN and temporal columns.

Example 8.2 - As an illustration on the sample database in subsection 8.1.3, we define two temporal structured types Emp_t and Dept_t with the life span support and two temporal typed tables Emp and Dept based on these two types, respectively.

```
CREATE TYPE Emp_t  WITH LIFE SPAN AS (
        Name            VARCHAR (64) VALIDTIME (DAY),
        Gender          CHAR(1),
        D_birth         DATE,
        Salary          NUMERIC VALIDTIME (MONTH));
CREATE TYPE Dept_t WITH LIFE SPAN AS (
        Department      VARCHAR (64),
        Manager         REF(Emp_t) VALIDTIME (MONTH),
        Budget          NUMERIC VALIDTIME (YEAR));
ALTER TYPE Emp_t
ADD ATTRIBUTE Dept REF(Dept_t) VALIDTIME (MONTH);
CREATE TABLE Dept OF Dept_t (
        REF IS REFC SYSTEM GENERATED,
        CONSTRAINT Dept_un UNIQUE (Department));
CREATE TABLE Emp OF Emp_t (
        REF IS REFC SYSTEM GENERATED,
        CONSTRAINT Name_un VALIDTIME UNIQUE(Name));
```

8.3.2.2 Temporal Database Schema Evolution Definitions

After the temporal aspect of columns of a table, the temporal aspect of attributes of a structured type, and integrity constraints on temporal columns of a temporal table are defined, a temporal database language should be capable of specifying the addition, removal or changes of these elements on demand. These requirements lead to several kinds of temporal evolutions at both type and table levels. So, our language extends the OR SQL language with temporal evolution definitions via the ALTER statements. More consideration on schema evolutions in object-oriented databases over the time has been given in [103, 104].

At the type level, the ALTER TYPE definition is enhanced with new alter type actions for the addition and dropping of a temporal attribute together with the addition and dropping of the temporal aspect of an attribute of a structured type. The addition of a temporal attribute is specified with the temporal attribute definition, presented in section 8.3.2.1. The dropping of a temporal attribute is the same as the dropping of a non-temporal attribute. The specification of an alter action for the addition and dropping of the temporal aspect of an attribute of a structured type is additionally defined with the new production rules <add_temporal_attribute_specification> and <drop_temporal_attribute_specification>. The addition of the temporal aspect of an attribute will turn a non-temporal structured type into a temporal structured type if the type has no temporal attribute before the alter type action. Thus, this action enables the temporal enhancement on a non-temporal database so that new applications can refer to temporal data while non-temporal applications run on the same database, which is now temporal, as if non-temporal before the alter type action. As a side effect, data

conversion of non-temporal attribute data to temporal attribute data and vice versa will be needed for existing contents of all typed tables defined on altered structured types. The enhanced ALTER TYPE statement syntax is shown below.

```
<tsql_alter_type_stmt> ::= ALTER TYPE <schema_resolved_tsql_user_defined_type_name> <alter_type_action>
<alter_type_action> ::= <add_attribute_definition> | <drop_attribute_definition> | <drop_method_specification>
        | <add_original_method_specification> | <add_overriding_method_specification>
        | <add_temporal_attribute_specification> | <drop_temporal_attribute_specification>
<add_temporal_attribute_specification> ::= MODIFY ATTRIBUTE <temporal_attribute_name>
                                           ADD <temporal_aspect>
<drop_temporal_attribute_specification> ::= MODIFY ATTRIBUTE <temporal_attribute_name>
                                           DROP VALIDTIME
```

At the table level, we consider only the addition and dropping of integrity constraints on temporal columns of a temporal table which is either a temporal 1NF/non-1NF table or a temporal typed table. Other alter table actions such as adding a column definition, dropping a column definition, and altering a column definition are only considered for a temporal 1NF/non-1NF table using the column definition in section 8.3.2.1. With the specification of a table constraint definition on temporal columns, ALTER TABLE statements are formed by using the conventional production rules of the non-temporal OR SQL language. All table constraint definitions related to temporal columns will minutely be presented in section 8.3.2.3. We allow integrity constraints on temporal columns to be added to a temporal table definition if and only if the existing content of the altered table satisfies all added constraints. If any added constraint does not hold on the existing content, the constraint addition is not accepted.

8.3.2.3 Integrity Constraint Definitions on Temporal Columns and Temporal Tables

In section 8.2, we have already defined inherent integrity constraints which implicitly hold on any temporal attribute of any temporal structured type and on any temporal column of any temporal table. In contrast, integrity constraints in this section are explicitly specified on temporal columns and their temporal tables in order to correctly reflect the universe of discourse. These explicit integrity constraints on temporal columns and temporal tables are derived from particular application requirements. They can be declared by database users in CREATE TABLE or ALTER TABLE statements.

Since object identifiers and references can be employed, the conventional primary key and referential integrity constraints are not discussed. Uniqueness constraints and

CHECK constraints are considered and generalized with respect to valid time. They are analogous to conventional non-temporal constraints at any instant [44]. In addition to the generalization of conventional non-temporal uniqueness and CHECK constraints, the notion of valid time homogeneity and the notion of valid time referential integrity constraints related to reference types are introduced. As compared to the existing works, valid time homogeneity constraints were not considered in [44, 105]. Therefore, we pay attention to four types of temporal integrity constraints, namely VALIDTIME HOMOGENEITY, VALIDTIME UNIQUE, VALIDTIME REFERENCES, and VALIDTIME CHECK. The syntax of these integrity constraint definitions on temporal columns and temporal tables is defined below by modifying the production rule <table_constraint> as <non_temporal_table_constraint> which is a conventional table constraint and <temporal_column_constraint_definition> which is a constraint on temporal columns.

<table_constraint> ::= <non_temporal_table_constraint> | <temporal_column_constraint_definition>
<temporal_column_constraint_definition> ::=
VALIDTIME HOMOGENEITY <left_paren> <temporal_column_name_list> <right_paren>
| VALIDTIME UNIQUE <left_paren> <temporal_column_name_list> <right_paren>
| REF ATTRIBUTE <left_paren> <temporal_column_name> <right_paren>
VALIDTIME REFERENCES <temporal_typed_table_name>
| VALIDTIME CHECK <left_paren> <search_condition> <right_paren>

Firstly, a VALIDTIME HOMOGENEITY constraint is flexibly allowed to hold so that the support of homogeneous tuples/rows of a temporal table can be up to particular application requirements to restrict the asynchronous behavior of temporal columns in a temporal table. Without the declaration of this constraint type, temporal columns are permitted to change over time independently of each other. The homogeneity is an option in our temporal data management in contrast to other related works whose models are either homogeneous [43] or heterogeneous [42] to database users.

Example 8.3 - To express the time-varying dependence of the salary and the department of an employee, a VALIDTIME HOMOGENEITY constraint needs to be defined as follows.

ALTER TABLE Emp
ADD CONSTRAINT Salary_Dept_hm **VALIDTIME HOMOGENEITY (Salary, Dept);**

Secondly, a valid time uniqueness constraint VALIDTIME UNIQUE is defined over a selected temporal column to specify the uniqueness of the value combination of these constrained columns over time so that no two tuples/rows of a temporal table have the same combination of values at the same time. This is a generalization of the conventional non-temporal uniqueness constraint with valid time. As shown in the

definition of the Emp temporal table, a valid time uniqueness constraint is specified on the Name temporal column to guarantee that no two employees have the same name at each common point in time.

Thirdly, a valid time referential integrity constraint VALIDTIME REFERENCES is innovatively defined as an analogy to non-temporal constraints on columns associated with attributes whose data types are reference types introduced in the OR data model. The valid time referential integrity constraint can be considered as referential integrity constraints in terms of object identifiers, references, and valid time.

Example 8.4 - To express the referential relationship between an employee and a department where the employee works for some periods of time, a valid time referential integrity constraint is defined. Such a constraint specifies the scope of a temporal attribute of a reference type.

```
ALTER TABLE Emp
ADD CONSTRAINT Dept_ref REF ATTRIBUTE (Dept) VALIDTIME REFERENCES Dept;
```

Fourthly, a VALIDTIME CHECK constraint is user-specified as a restriction on each individual tuple/row of a temporal table by a particular condition on at least one temporal column. A VALIDTIME CHECK condition is in fact a search condition that we will take into account in section 8.3.3.2.

Example 8.5 - A VALIDTIME CHECK constraint is defined below on the Salary temporal column of the Emp temporal table. It specifies that the salary of any employee is always greater than 15.

```
ALTER TABLE Emp
ADD CONSTRAINT Salary_chk VALIDTIME CHECK (ALWAYS {Salary>15});
```

For temporal integrity constraint checking which cannot be processed by an employed non-temporal DBMS, all constraints are translated into respective temporal queries with their negated conditions [44]. At commit time of any modification operation (insertion, deletion, and update), relevant queries are executed. The result will determine whether or not the modification operation violates any constraint. If yes, the modification operation will be rejected. Otherwise, it will be accepted.

8.3.3 Temporal Data Querying

Temporal data querying with the valid time support at the attribute level is presented in this section. It is associated with temporal logic [21] by employing temporal operators ALWAYS, ANYTIME, NEXT, PREV, and UNTIL. For data retrieval from temporal tables ut_1, ..., ut_k, ..., and ut_z, we examine the following form of a SELECT statement where the ORDER BY clause is ignored. More complex queries can be achieved from

the further refinement of the results of less complex queries or from the direct use of our temporal methods within non-sequenced statements.

SELECT [<set_quantifier>] <select_list>	(8.3.3.3)
FROM <table_reference_list>	(8.3.3.1)
[<where_clause>]	(8.3.3.2)
[<group_by_clause>]	(8.3.3.4)
[<having_clause>]	(8.3.3.4)

Before each clause of a SELECT statement is detailed, it is noted that in our language, attribute histories returned by temporal column references are treated as first class values that can be queried and manipulated as the whole because temporal column references are in fact column references. In turn, column references can be present in value expressions that either are returned to database users or participate in some search condition for joins and selections. From this point, each clause supported in our language is considered with the temporal aspect of input tables, in fact of involved columns. Besides, the Cartesian product (i.e. CROSS JOIN), union (UNION), difference (EXCEPT in [17] or MINUS in Oracle 10g/11g), and intersection (INTERSECT) on temporal tables are the same as those on non-temporal tables. It is because these operations perform at the object/tuple level and they are not affected by the temporal aspect of any column of an input table. Therefore, temporal tables become non-temporal ones with regard to these operations and the union compatibility checking is performed on temporal columns as usual.

8.3.3.1 FROM Clause

In the FROM clause, a <table_reference_list> is used to derive a list of table references. In case of a list of two or many table references, the Cartesian product, i.e. CROSS JOIN, is obtained as a result of the evaluation of this clause. The operation at this stage is performed at the object/tuple level. However, the specification of the joins different from a cross join might be placed in this clause according to the specification of the OR SQL language in [17]. That could be a qualified join or a natural join.

- A qualified join: in the join specification, a search condition is defined with the ON keyword or a join column list is determined with the USING keyword. So, a qualified join is said to be temporal if any temporal column reference is involved in the join specification. Otherwise, non-temporal.

- A natural join: a common equi-join column list is implicitly determined. All common referenced columns must be comparable and operands of an equality operation. Similarly, a natural join is said to be temporal if at least one common referenced column is temporal. Otherwise, non-temporal.

In general, a join operation among tables can be expressed by a combination of the Cartesian product in the FROM clause and appropriate search conditions in the WHERE clause instead of explicitly using join operators: JOIN and NATURAL JOIN. To simplify and minimize our temporal extension, a join operation on temporal tables is explicitly formed by database users with the Cartesian product in the FROM clause and search conditions in the WHERE clause. So, the FROM clause in our language is simply a list of table references that are table names with/without their aliases separated from each other by commas. Its syntax is specified as follows.

<table_reference_list> ::= <table_reference> | <table_reference_list> <comma> <table_reference>
<table_reference> ::= <table_name> [[AS] <table_alias>]

8.3.3.2 WHERE Clause

Consider the WHERE clause: <where_clause> ::= WHERE <search_condition>.

If the WHERE clause is not specified, all rows of a resultant table from the evaluation of the FROM clause are returned as a result of the evaluation of the WHERE clause.

If the WHERE clause is specified, the main issue is the specification of a <search_condition> in respect of the temporal aspect of columns involved in a <search_condition>. According to SQL production rules, a <search_condition> is shaped by one or many so-called Boolean primaries with or without Boolean connectives such as NOT, AND, and OR together with the grouping parentheses '(' and ')'. The existing SQL specification defines a Boolean primary as a predicate such as a comparison predicate, a BETWEEN predicate, an IN predicate, a LIKE predicate, a NULL predicate, etc. or as a Boolean predicand such as a parenthesized Boolean value expression or a nonparenthesized value expression primary. We pay attention to so-called predicates defined in [17]. These predicates are primitive in such a way that any of them can not be further decomposed into other predicates. The interpretation of each predicate will assign a Boolean value (TRUE or FALSE) as the meaning of that predicate

at the evaluation time. Besides, predicands of each predicate are row value predicands which are expressions in which column references might be present. Thus, the specification of a <search_condition> is now shifted to the specification of a predicate with respect to the consideration on whether any column reference specified in the predicate specification is related to any temporal column.

- If no column reference is related to any temporal column, the predicate is interpreted as usual.

- If at least one column reference is related to a temporal column, the predicate is interpreted according to the time-varying aspect of the referenced temporal column. The time-varying of referenced temporal columns makes the predicate temporal as a temporal formula of temporal logic. Indeed, the truth value of the predicate changes over time along with the change of referenced temporal columns. So, the employment of temporal logic [21] to describe how the truth value of a predicate changes by means of typical temporal operators ALWAYS, ANYTIME (EVENTUALLY), NEXT, PREV (LAST), and UNTIL is appropriate. We consider whether or not the temporal aspect is expected.

i – The temporal aspect is not expected. The principle of temporal upward compatibility is applied to the interpretation of the predicate which is being evaluated. In other words, only current states of temporal columns which take part in the predicate are used. The predicate is regarded to be time-invariant.

Example 8.6 – A non-temporal (current) query on the Emp temporal table without any consideration on the temporal aspect of the Dept temporal column: List employees who (now) work in the Sales department.

```
SELECT e.REFC AS ID
FROM Emp e
WHERE e.Dept.Department = 'Sales';
```

ID
E4

ii – The temporal aspect is expected. In order to express the time-varying aspect of a predicate containing temporal column references, we extend the set of Boolean primaries with temporal predicates TP as shown below. A temporal predicate is a predicate whose truth value changes along time. The specification of a temporal predicate is formed by adding a temporal operator into the specification of a predicate defined in [17] so that conventional predicates become arguments of temporal operators. Temporal operators which are based on temporal logic are ALWAYS, ANYTIME, NEXT, PREV, and UNTIL along the time line (past, NOW, and future). Other

temporal operators such as SINCE can be derived from these typical temporal operators. Each temporal predicate is also interpreted as a truth value (TRUE or FALSE). It can be used wherever a standard OR SQL predicate is allowed in a search condition.

<boolean_primary> ::= <predicate> | <boolean_predicand> | <temporal_predicate>
<temporal_predicate> ::= ALWAYS <left_bracket> <predicate> <right_bracket> (TP1)
| ALWAYS <periods> <left_bracket> <predicate> <right_bracket> (TP2)
| pALWAYS <left_bracket> <predicate> <right_bracket> (TP3)
| fALWAYS <left_bracket> <predicate> <right_bracket> (TP4)
| ANYTIME <left_bracket> <predicate> <right_bracket> (TP5)
| ANYTIME <periods> <left_bracket> <predicate> <right_bracket> (TP6)
| pANYTIME <left_bracket> <predicate> <right_bracket> (TP7)
| fANYTIME <left_bracket> <predicate> <right_bracket> (TP8)
| NEXT <left_bracket> <predicate> <right_bracket> (TP9)
| PREV <left_bracket> <predicate> <right_bracket> (TP10)
| UNTIL <left_bracket> <predicate> <comma> <predicate> <right_bracket> (TP11)
| UNTIL <periods> <left_bracket> <predicate> <comma> <predicate> <right_bracket> (TP12)
| pUNTIL <left_bracket> <predicate> <comma> <predicate> <right_bracket> (TP13)
| fUNTIL <left_bracket> <predicate> <comma> <predicate> <right_bracket> (TP14)

Let p and q be two predicates shaped by production rules in [17] with none of our new temporal constructs.

Let tc_1^p, ..., tc_i^p, ..., and tc_n^p be temporal column references present in p and their corresponding attribute histories be $tc_1^p = \{(av_{1,1}^p, vt_{1,1}^p), ..., (av_{1,c1}^p, vt_{1,c1}^p), ..., (av_{1,h1}^p, vt_{1,h1}^p)\}$, ..., $tc_i^p = \{(av_{i,1}^p, vt_{i,1}^p), ..., (av_{i,ci}^p, vt_{i,ci}^p), ..., (av_{i,hi}^p, vt_{i,hi}^p)\}$, ..., and $tc_n^p = \{(av_{n,1}^p, vt_{n,1}^p), ..., (av_{n,cn}^p, vt_{n,cn}^p), ..., (av_{n,hn}^p, vt_{n,hn}^p)\}$.

Let tc_1^q, ..., tc_i^q, ..., and tc_m^q be temporal column references present in q and their corresponding attribute histories be $tc_1^q = \{(av_{1,1}^q, vt_{1,1}^q), ..., (av_{1,c1}^q, vt_{1,c1}^q), ..., (av_{1,h1}^q, vt_{1,h1}^q)\}$, ..., $tc_i^q = \{(av_{i,1}^q, vt_{i,1}^q), ..., (av_{i,ci}^q, vt_{i,ci}^q), ..., (av_{i,hi}^q, vt_{i,hi}^q)\}$,..., and $tc_m^q = \{(av_{m,1}^q, vt_{m,1}^q), ..., (av_{m,cm}^q, vt_{m,cm}^q), ..., (av_{m,hm}^q, vt_{m,hm}^q)\}$.

We elaborately formulate the sequenced semantics of these temporal predicates.

Firstly, we define the time line over which a temporal predicate is evaluated. The time line is discrete. It is partitioned into granules based on a common granularity and open to both the past and future. It contains a special instant, NOW, for the separation of the past and future. The smallest instant INFINITY- is the infinity towards the past and the greatest instant INFINITY+ is the one towards the future. NOW+1 is the next instant of NOW towards the future and NOW-1 the previous instant of NOW towards the past.

Secondly, we define the meaning of truth of each group of temporal predicates TP1...TP14. For clarity, an illustration on the evaluation of these temporal operators with the presence of two temporal column references, namely x1 and x2, is shown as follows in Figures 8.2..8.6.

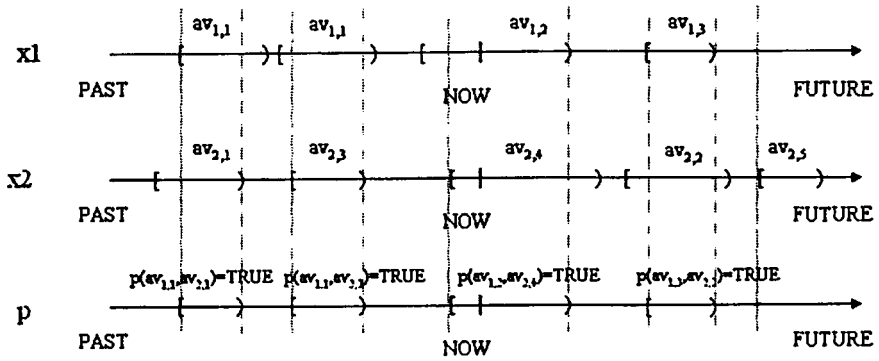


Figure 8.2 The Evaluation of the ALWAYS Temporal Operators: ALWAYS {<p(x1, x2)>}

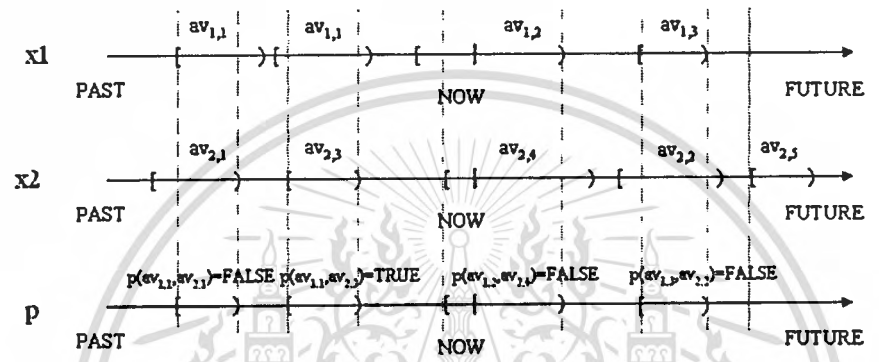


Figure 8.3 The Evaluation of the ANYTIME Temporal Operators: ANYTIME {<p(x1, x2)>}

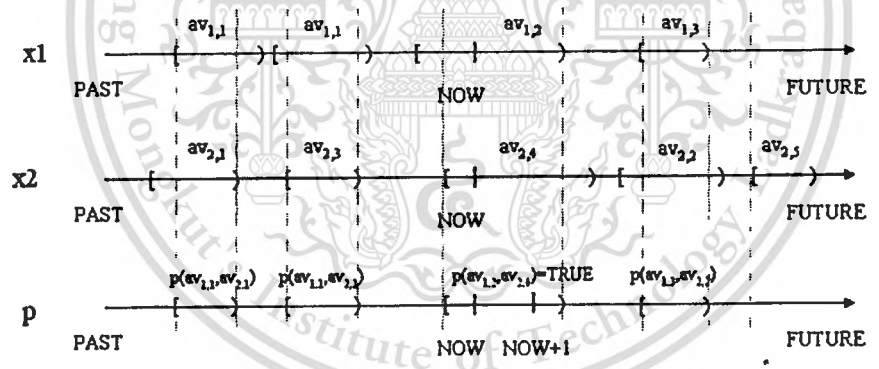


Figure 8.4 The Evaluation of the NEXT Temporal Operators: NEXT {<p(x1, x2)>}

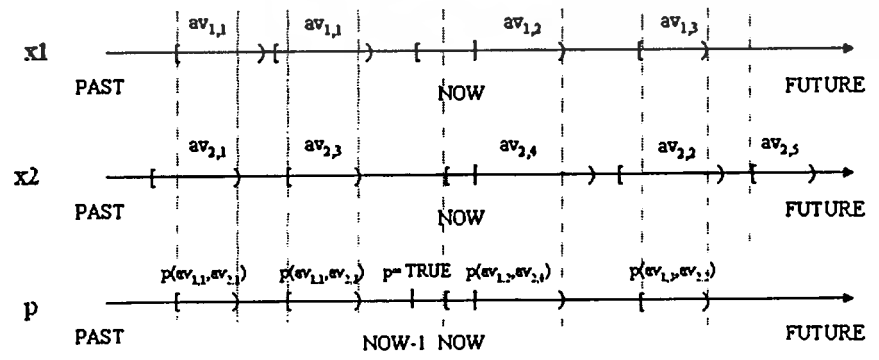


Figure 8.5 The Evaluation of the PREV Temporal Operators: PREV {<p(x1, x2)>}

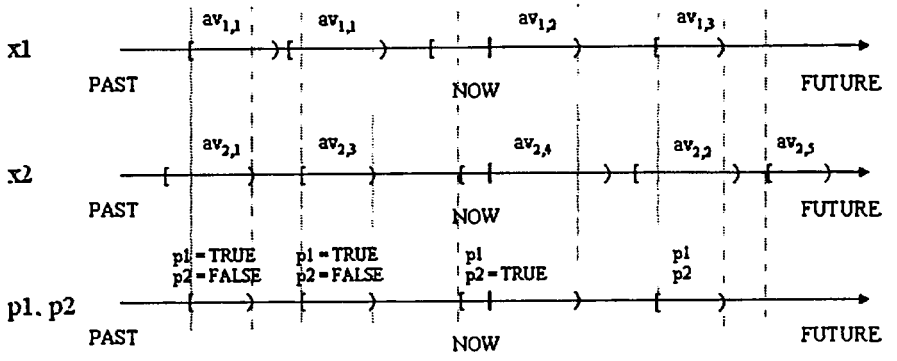


Figure 8.6 The Evaluation of the UNTIL Temporal Operators:

UNTIL {<p1(x1, x2)>, <p2(x1, x2)>}

TP1 – ALWAYS $\{p(tc_1^P, ..., tc_i^P, ..., tc_n^P)\}$ = TRUE if and only if
$\forall\ c1^P$ in 1..h1, ..., ci^P in 1..hi, ..., cn^P in 1..hn,
$vt_{1,c1}^P \cap ... \cap vt_{i,ci}^P \cap ... \cap vt_{n,cn}^P \neq \emptyset$ and $p(av_{1,c1}^P, ..., av_{i,ci}^P, ..., av_{n,cn}^P)$ = TRUE.

TP2 – ALWAYS <periods>$\{p(tc_1^P, ..., tc_i^P, ..., tc_n^P)\}$ = TRUE if and only if
$\forall\ c1^P$ in 1..h1, ..., ci^P in 1..hi, ..., cn^P in 1..hn,
$vt_{1,c1}^P \cap ... \cap vt_{i,ci}^P \cap ... \cap vt_{n,cn}^P \cap$ <periods> $\neq \emptyset$ and $p(av_{1,c1}^P, ..., av_{i,ci}^P, ..., av_{n,cn}^P)$ = TRUE.

TP3 – pALWAYS $\{p(tc_1^P, ..., tc_i^P, ..., tc_n^P)\}$ = TRUE if and only if
$\forall\ c1^P$ in 1..h1, ..., ci^P in 1..hi, ..., cn^P in 1..hn,
$vt_{1,c1}^P \cap ... \cap vt_{i,ci}^P \cap ... \cap vt_{n,cn}^P \cap \{[INFINITY-, NOW)\} \neq \emptyset$ and $p(av_{1,c1}^P, ..., av_{i,ci}^P, ..., av_{n,cn}^P)$ = TRUE.

TP4 – fALWAYS $\{p(tc_1^P, ..., tc_i^P, ..., tc_n^P)\}$ = TRUE if and only if
$\forall\ c1^P$ in 1..h1, ..., ci^P in 1..hi, ..., cn^P in 1..hn,
$vt_{1,c1}^P \cap ... \cap vt_{i,ci}^P \cap ... \cap vt_{n,cn}^P \cap \{[NOW, INFINITY+)\} \neq \emptyset$ and $p(av_{1,c1}^P, ..., av_{i,ci}^P, ..., av_{n,cn}^P)$ = TRUE.

TP5 – ANYTIME $\{p(tc_1^P, ..., tc_i^P, ..., tc_n^P)\}$ = TRUE if and only if
$\exists\ c1^P$ in 1..h1, ..., ci^P in 1..hi, ..., cn^P in 1..hn,
$vt_{1,c1}^P \cap ... \cap vt_{i,ci}^P \cap ... \cap vt_{n,cn}^P \neq \emptyset$ and $p(av_{1,c1}^P, ..., av_{i,ci}^P, ..., av_{n,cn}^P)$ = TRUE.

TP6 – ANYTIME <periods> $\{p(tc_1^P, ..., tc_i^P, ..., tc_n^P)\}$ = TRUE if and only if
$\exists\ c1^P$ in 1..h1, ..., ci^P in 1..hi, ..., cn^P in 1..hn,
$vt_{1,c1}^P \cap ... \cap vt_{i,ci}^P \cap ... \cap vt_{n,cn}^P \cap$ <periods> $\neq \emptyset$ and $p(av_{1,c1}^P, ..., av_{i,ci}^P, ..., av_{n,cn}^P)$ = TRUE.

TP7 – pANYTIME $\{p(tc_1^P, ..., tc_i^P, ..., tc_n^P)\}$ = TRUE if and only if
$\exists\ c1^P$ in 1..h1, ..., ci^P in 1..hi, ..., cn^P in 1..hn,
$vt_{1,c1}^P \cap ... \cap vt_{i,ci}^P \cap ... \cap vt_{n,cn}^P \cap \{[INFINITY-, NOW)\} \neq \emptyset$ and $p(av_{1,c1}^P, ..., av_{i,ci}^P, ..., av_{n,cn}^P)$ = TRUE.

TP8 – fANYTIME $\{p(tc_1^P, ..., tc_i^P, ..., tc_n^P)\}$ = TRUE if and only if
$\exists\ c1^P$ in 1..h1, ..., ci^P in 1..hi, ..., cn^P in 1..hn,
$vt_{1,c1}^P \cap ... \cap vt_{i,ci}^P \cap ... \cap vt_{n,cn}^P \cap \{[NOW, INFINITY+)\} \neq \emptyset$ and $p(av_{1,c1}^P, ..., av_{i,ci}^P, ..., av_{n,cn}^P)$ = TRUE.

TP9 – NEXT $\{p(tc_1^P, ..., tc_i^P, ..., tc_n^P)\}$ = TRUE if and only if
$\exists\ c1^P$ in 1..h1, ..., ci^P in 1..hi, ..., cn^P in 1..hn,
NOW+1 $\in vt_{1,c1}^P$ and ... and NOW+1 $\in vt_{i,ci}^P$ and ... and NOW+1 $\in vt_{n,cn}^P$
and $p(av_{1,c1}^P, ..., av_{i,ci}^P, ..., av_{n,cn}^P)$ = TRUE.

TP10 – PREV $\{p(tc_1^P, ..., tc_i^P, ..., tc_n^P)\}$ = TRUE if and only if
$\exists\ c1^P$ in 1..h1, ..., ci^P in 1..hi, ..., cn^P in 1..hn,
NOW-1 $\in vt_{1,c1}^P$ and ... and NOW-1 $\in vt_{i,ci}^P$ and ... and NOW-1 $\in vt_{n,cn}^P$
and $p(av_{1,c1}^P, ..., av_{i,ci}^P, ..., av_{n,cn}^P)$ = TRUE.

TP11 – UNTIL $\{p(tc_1^P, ..., tc_i^P, ..., tc_n^P), q(tc_1^q, ..., tc_i^q, ..., \text{and } tc_m^q)\}$ = TRUE if and only if
$\exists\ c1^q$ in 1..h1, ..., ci^q in 1..hi, ..., cm^q in 1..hm, qt in $vt_{1,c1}^q \cap ... \cap vt_{i,ci}^q \cap ... \cap vt_{m,cm}^q$,
qt != NULL and $q(av_{1,c1}^q, ..., av_{i,ci}^q, ..., av_{m,cm}^q)$ = TRUE
and ($\forall\ c1^P$ in 1..h1, ..., ci^P in 1..hi, ..., cn^P in 1..hn,
$vt_{1,c1}^P \cap ... \cap vt_{i,ci}^P \cap ... \cap vt_{n,cn}^P \cap \{[INFINITY-, qt)\} \neq \emptyset$ and $p(av_{1,c1}^P, ..., av_{i,ci}^P, ..., av_{n,cn}^P)$ = TRUE).

TP12 – UNTIL <periods> $\{p(tc_1^P, ..., tc_i^P, ..., tc_n^P), q(tc_1^q, ..., tc_i^q, ..., \text{and } tc_m^q)\}$ = TRUE if and only if
$\exists\ c1^q$ in 1..h1, ..., ci^q in 1..hi, ..., cm^q in 1..hm, qt in $vt_{1,c1}^q \cap ... \cap vt_{i,ci}^q \cap ... \cap vt_{m,cm}^q \cap$ <periods>,
qt != NULL and $q(av_{1,c1}^q, ..., av_{i,ci}^q, ..., av_{m,cm}^q)$ = TRUE

and ($\forall$ $c1^p$ in 1..h1, ..., ci^p in 1..hi, ..., cn^p in 1..hn,
$vt_{1,c1}{}^p \cap \ldots \cap vt_{i,ci}{}^p \cap \ldots \cap vt_{n,cn}{}^p \cap \{[INFINITY-, qt)\} != \emptyset$ and $p(av_{1,c1}{}^p, \ldots, av_{i,ci}{}^p, \ldots, av_{n,cn}{}^p)$ = TRUE).

TP13 – pUNTIL $\{p(tc_1{}^p, \ldots, tc_i{}^p, \ldots, tc_n{}^p), q(tc_1{}^q, \ldots, tc_i{}^q, \ldots, \text{and } tc_m{}^q)\}$ = TRUE if and only if
$\exists$ $c1^q$ in 1..h1,...,ci^q in 1..hi,...,cm^q in 1..hm,qt in $vt_{1,c1}{}^q \cap \ldots \cap vt_{i,ci}{}^q \cap \ldots \cap vt_{m,cm}{}^q \cap \{[INFINITY-, NOW)\}$,
qt != NULL and $q(av_{1,c1}{}^q, \ldots, av_{i,ci}{}^q, \ldots, av_{m,cm}{}^q)$ = TRUE
and ($\forall$ $c1^p$ in 1..h1, ..., ci^p in 1..hi, ..., cn^p in 1..hn,
$vt_{1,c1}{}^p \cap \ldots \cap vt_{i,ci}{}^p \cap \ldots \cap vt_{n,cn}{}^p \cap \{[INFINITY-, qt)\} != \emptyset$ and $p(av_{1,c1}{}^p, \ldots, av_{i,ci}{}^p, \ldots, av_{n,cn}{}^p)$ = TRUE).

TP14 – fUNTIL $\{p(tc_1{}^p, \ldots, tc_i{}^p, \ldots, tc_n{}^p), q(tc_1{}^q, \ldots, tc_i{}^q, \ldots, \text{and } tc_m{}^q)\}$ = TRUE if and only if
$\exists$ $c1^q$ in 1..h1,...,ci^q in 1..hi,...,cm^q in 1..hm,qt in $vt_{1,c1}{}^q \cap \ldots \cap vt_{i,ci}{}^q \cap \ldots \cap vt_{m,cm}{}^q \cap \{[NOW, INFINITY+)\}$,
qt != NULL and $q(av_{1,c1}{}^q, \ldots, av_{i,ci}{}^q, \ldots, av_{m,cm}{}^q)$ = TRUE
and ($\forall$ $c1^p$ in 1..h1, ..., ci^p in 1..hi, ..., cn^p in 1..hn,
$vt_{1,c1}{}^p \cap \ldots \cap vt_{i,ci}{}^p \cap \ldots \cap vt_{n,cn}{}^p \cap \{[INFINITY-, qt)\} != \emptyset$ and $p(av_{1,c1}{}^p, \ldots, av_{i,ci}{}^p, \ldots, av_{n,cn}{}^p)$ = TRUE).

Finally, we intuitively state the meaning of truth of each group of temporal predicates TP1...TP14.

TP1 – ALWAYS {p} ≡ p is always true along the time line stretching the past, present, and future.
TP2 – ALWAYS <periods> {p} ≡ p is always true for some periods of time.
TP3 – pALWAYS {p} ≡ p is always true in the past.
TP4 – fALWAYS {p} ≡ p is always true in the future.
TP5 – ANYTIME {p} ≡ p is true sometime along the time line stretching the past, present, and future.
TP6 – ANYTIME <periods> {p} ≡ p is true sometime for some periods of time.
TP7 – pANYTIME {p} ≡ p is true sometime in the past.
TP8 – fANYTIME {p} ≡ p is true sometime in the future.
TP9 – NEXT {p} ≡ p is true next time with a common granularity of all referenced temporal columns.
TP10 – PREV {p} ≡ p is true last time with a common granularity of all referenced temporal columns.
TP11 – UNTIL {p, q} ≡ q is true sometime along the time line and p is true till that moment.
TP12 – UNTIL<periods>{p, q} ≡ q is true sometime for some time periods and p is true till that moment.
TP13 – pUNTIL {p, q} ≡ q is true sometime in the past and p is true till that moment.
TP14 – fUNTIL {p, q} ≡ q is true sometime in the future and p is true till that moment.

For illustration, we present a few examples of SELECT statements on the two Emp and Dept temporal tables of the sample database. Examples 8.13..8.16 are temporal SELECT statements with join conditions.

Example 8.7 – A temporal query on the Emp temporal table using the ALWAYS temporal operator for a selection condition and the ASELECT construct which is defined in the next subsection for data retrieval related to attribute histories of some data type specified with the clause "OF <type_name>": What are the names and salaries of those employees in the Sales department during 1990-1995?

```
SELECT e.REFC AS ID, ASELECT PERIOD('01-01-1990', '01-01-1996') {e.Name OF VARCHAR} AS NAME,
                     ASELECT PERIOD('01-01-1990', '01-01-1996') {e.Salary OF NUMERIC} AS SALARY
FROM Emp e
WHERE ALWAYS PERIOD('01-01-1990', '01-01-1996') {e.Dept.Department = 'Sales'};
```

ID	NAME			SALARY		
	vtValue	ValidTime		vtValue	ValidTime	
		vtStart	vtEnd		vtStart	vtEnd
E4	Jack	1/1/1990	1/1/1996	50	1/1/1990	1/1/1996

Example 8.8 – A temporal query with the pALWAYS past temporal operator: Who are now the managers of the departments whose budgets are currently less than 500 and were always greater than 100?

```
SELECT d.REFC AS ID, d.Manager AS MANAGER
FROM Dept d
WHERE d.Budget < 500 AND pALWAYS {d.Budget > 100};
```

ID	MANAGER
D2	E1
D3	E4
D4	E3

Example 8.9 – A temporal query with the pANYTIME past temporal operator: What are the name and salary histories of those employees who worked sometime in the Sales department in the past?

```
SELECT e.REFC AS ID, ASELECT{e.Name OF VARCHAR} AS NAME, ASELECT{e.Salary OF NUMERIC} AS SALARY
FROM Emp e
WHERE pANYTIME {e.Dept.Department = 'Sales'};
```

ID	NAME			SALARY		
	vtValue	ValidTime		vtValue	ValidTime	
		vtStart	vtEnd		vtStart	vtEnd
E4	Jack	5/15/1950	12/31/9999	40	1/1/1980	1/1/1984
				50	1/1/1984	12/31/9999
E5	White	12/1/1960	12/31/9999	30	1/1/1980	1/1/1984
				40	1/1/1984	1/1/1989
				50	1/1/1989	12/31/9999

Example 8.10 – A temporal query with the PREV temporal operator: List employees whose salary of last month was greater than 40.

```
SELECT e.REFC AS ID
FROM Emp e
WHERE PREV {e.Salary > 40};
```

ID
E2
E3
E4
E5

Example 8.11 – A temporal query with the NEXT temporal operator: List employees who are going to move to the Sales department next month.

```
SELECT e.REFC AS ID
FROM Emp e
WHERE NEXT {e.Dept.Department = 'Sales'};
```

ID
E4

Example 8.12 – A temporal query with the pUNTIL past temporal operator: What are the name and salary histories of those employees who worked only in the Sales department until changing to the Marketing department in the past?

```
SELECT e.REFC AS ID, ASELECT{e.Name OF VARCHAR} AS NAME, ASELECT{e.Salary OF NUMERIC} AS SALARY
FROM Emp e
WHERE pUNTIL {e.Dept.Department = 'Sales', e.Dept.Department = 'Marketing'};
```

ID	NAME			SALARY		
	vtValue	ValidTime		vtValue	ValidTime	
		vtStart	vtEnd		vtStart	vtEnd
E5	White	12/1/1960	12/31/9999	30	1/1/1980	1/1/1984
				40	1/1/1984	1/1/1989
				50	1/1/1989	12/31/9999

Example 8.13 - A non-temporal join on the Emp temporal table: List employees who have the same salary.

```
SELECT e1.REFC AS ID1, e2.REFC AS ID2
FROM Emp e1, Emp e2
WHERE e1.REFC < e2.REFC AND e1.Salary = e2.Salary;
```

ID1	ID2
E2	E4

Example 8.14 - A temporal join using the ALWAYS temporal operator on the Emp temporal table: List all employees who have had the same salary for Feb 1985-Feb 1986.

```
SELECT e1.REFC AS ID1, e2.REFC AS ID2
FROM Emp e1, Emp e2
WHERE e1.REFC < e2.REFC
AND ALWAYS PERIOD('02-01-1985', '03-01-1986') {e1.Salary = e2.Salary};
```

ID1	ID2
E1	E2
E1	E5
E2	E5

Example 8.15 - A temporal join using the pUNTIL past temporal operator: List departments which always got the same budget as the Sales department until the Sales department got a greater amount.

```
SELECT d1.REFC AS ID, d1.Department AS DEPARTMENT
FROM Dept d1, Dept d2
WHERE d1.REFC <> d2.REFC AND d2.Department = 'Sales'
AND pUNTIL {d1.Budget = d2.Budget, d1.Budget < d2.Budget};
```

ID	DEPARTMENT
D1	Toy

Example 8.16 - A temporal join with the fALWAYS and pANYTIME temporal operators: List employees who always work in the same department from now but got different salaries sometime in the past.

```
SELECT e1.REFC AS ID1, e2.REFC AS ID2
FROM Emp e1, Emp e2
WHERE e1.REFC < e2.REFC
AND fALWAYS {e1.Dept = e2.Dept}
AND pANYTIME {e1.Salary != e2.Salary};
```

ID1	ID2
E3	E5

Although we allow the temporal operators to be applied to predicates, the following distributivity relations between the temporal operators and the Boolean connectives (NOT, AND, and OR) can be utilized for more complex temporal requirements. Let p, q, and r be conventional predicates.

```
ALWAYS {p AND q} ≡ ALWAYS {p} AND ALWAYS {q};
ALWAYS {NOT p} ≡ NOT ANYTIME {p};
ANYTIME {NOT p} ≡ NOT ALWAYS {p};
ANYTIME {p OR q} ≡ ANYTIME {p} OR ANYTIME {q};
NEXT {p OR q} ≡ NEXT {p} OR NEXT {q};
NEXT {p AND q} ≡ NEXT {p} AND NEXT {q};
NEXT {NOT p} ≡ NOT NEXT {p};
PREV {p AND q} ≡ PREV {p} AND PREV {q};
PREV {p OR q} ≡ PREV {p} OR PREV {q};
PREV {NOT q} ≡ NOT PREV {q};
UNTIL {p AND q, r} ≡ UNTIL {p, r} AND UNTIL {q, r};
UNTIL {p, q OR r} ≡ UNTIL {p, q} OR UNTIL {p, r}.
```

It is noted that time variables disappear in temporal logic. This fact reduces the great difficulty and complexity of expressing the change of truth values of assertions over time as compared to the use of pure first order logic with time variables. Hence, it is valuable to take advantage of temporal logic for the temporal data querying support of the proposed language. Using our language with the expressive power of temporal logic, database users can easily and naturally issue temporal queries by means of temporal operators as demonstrated in the examples.

8.3.3.3 SELECT Clause

For a temporal extension in the SELECT clause of a query statement, we examine a <select_list> with consideration on the temporal aspect of involved column references. A <select_list> comprises a collection of <select_sublist>s separated from each other

by commas. The production rules of a <select_sublist> are modified below to include the derivation of temporal columns returned to database users:

<select_sublist> ::= (<derived_non_temporal_column> | <derived_temporal_column>) [<as_clause>]

A <select_sublist> is either a <derived_non_temporal_column> or a <derived_temporal_column>.

A <derived_non_temporal_column> is in fact a conventional <derived_column> of the non-temporal OR SQL language [17]. In this case, no temporal aspect is concerned. A derived column can be any value expression defined in the non-temporal OR SQL language or in the core of SQL:2003, implemented in any existing ORDBMS. So, the evaluation of a derived non-temporal column is checked with regard to the presence of any temporal column reference.

If there is no presence of any temporal column reference, no temporal aspect exists. The evaluation of the derived non-temporal column being considered is usual.

If there are one or many presences of temporal column references, the temporal aspect exists, but is not desired. The derived column is evaluated with current states of temporal columns. These temporal columns are treated as if non-temporal according to the principle of temporal upward compatibility.

Example 8.17 - We want to know the current name of each employee. The Name temporal attribute is treated as if it were a non-temporal one.

```
SELECT e.REFC AS ID, e.Name AS NAME
FROM Emp e;
```

ID	NAME
E1	Edward
E2	Di
E3	Johnson
E4	Jack
E5	White

Different from a <derived_non_temporal_column>, a <derived_temporal_column> is defined below.

<derived_temporal_column> ::= ASELECT [<periods>] (S3)
<left_bracket> <value_expression> OF <type_name> (S1)
[ON <search_condition>] <right_bracket> (S2)

The ASELECT construct provides the notion of history oriented selection examined on one or many attribute histories of one or many time-varying elementary facts. This notion is comparable to the notion of valid time selection [19] on temporal 1NF relations to extract facts from a temporal database based on the relational data model. In our work, an expected value at a derived temporal column returned to temporal database

users is a collection of temporal compositions defined in section 8.2. Each temporal composition is computed from one or many attribute histories referred to by temporal column references corresponding to temporal elementary facts gathered in a row/tuple of a resultant temporal table after the evaluation of the FROM and WHERE clauses. For the evaluation of a <derived_temporal_column>, we consider three main parts of a <derived_temporal_column>, namely, <value_expression> OF <type_name>, ON <search_condition>, and <periods>. These three parts will be detailed as follows in the semantics explanations S1, S2, and S3, respectively.

S1 – <value_expression> OF <type_name>

The specification of a <value_expression> is a conventional one without any of our new temporal constructs in the OR SQL language [17] or in the core of SQL:2003, implemented in an ORDBMS.

If ASELECT wraps no column reference that refers to an attribute history at a temporal column, sequenced semantics has no impact on the returned value. The evaluation of a derived temporal column is the same as the one of a derived non-temporal column without the ASELECT construct.

If one or many attribute histories involve in a <value_expression> via one or many column references which are now called temporal column references, their temporal aspect is desired to be considered. Each <value_expression> is specified along with the name <type_name> of its data type, which can be a built-in type (e.g. VARCHAR, NUMERIC, etc.), a DBMS-supplied type (e.g. MDSYS.SDO_GEOMETRY on Oracle 10g/11g), a user-defined type (e.g. T3D.TYPE1, which is a user-defined type under the T3D schema), or a REF type (e.g. REF(Dept_t)), etc. This data type will be the data type of the time-dependent component of every temporal composition of the resultant collection at a derived temporal column.

Let $\{(av_{j,1}, vt_{j,1}), \ldots, (av_{j,c}, vt_{j,c}), \ldots, (av_{j,h}, vt_{j,h})\}$ be an expected result after the evaluation S1.

Let $tc_1, \ldots, tc_i, \ldots, tc_n$ be temporal column references in a <value_expression> and their corresponding attribute histories be $tc_1 = \{(av_{1,1}, vt_{1,1}), \ldots, (av_{1,c1}, vt_{1,c1}), \ldots, (av_{1,h1}, vt_{1,h1})\}$,

..., tc_i = {($av_{i,1}$, $vt_{i,1}$), ..., ($av_{i,ci}$, $vt_{i,ci}$), ..., ($av_{i,hi}$, $vt_{i,hi}$)}, ..., and tc_n = {($av_{n,1}$, $vt_{n,1}$), ..., ($av_{n,cn}$, $vt_{n,cn}$), ..., ($av_{n,hn}$, $vt_{n,hn}$)}.

Let F be a final function on these temporal column references that returns a scalar value of the data type <type_name>. We can rewrite a <value_expression> as F(tc_1, ..., tc_i, ..., tc_n). For example, <value_expression> = function1(tc1 + tc2, tc1) – 2*tc3 = F(tc1, tc2, tc3).

The evaluation S1 of a <value_expression> with sequenced semantics is shown below to compute one temporal composition ($av_{j,c}$, $vt_{j,c}$) in the result of the evaluation S1.

∀ c1 in 1..h1, ..., ci in 1..hi, ..., cn in 1..hn,
if $vt_{1,c1} \cap \ldots \cap vt_{i,ci} \cap \ldots \cap vt_{n,cn} \neq \emptyset$ and $F(av_{1,c1}, \ldots, av_{i,ci}, \ldots, av_{n,cn}) \neq$ NULL
then for c in 1..h, $av_{j,c} = F(av_{1,c1}, \ldots, av_{i,ci}, \ldots, av_{n,cn})$; $vt_{j,c} = vt_{1,c1} \cap \ldots \cap vt_{i,ci} \cap \ldots \cap vt_{n,cn}$.

Example 8.18 - We can think of the whole history of something/someone, e.g. the history of each employee's name.

```
SELECT REFC AS ID, ASELECT {Name OF VARCHAR} AS NAME
FROM Emp;
```

ID	NAME		
	vtValue	ValidTime	
		vtStart	vtEnd
E1	Ed	2/1/1982	1/1/1988
	Edward	1/1/1988	12/31/9999
E2	Di	1/1/1982	12/31/9999
E3	John	1/1/1962	1/1/1978
	Johnson	1/1/1978	12/31/9999
E4	Jack	5/15/1950	12/31/9999
E5	White	12/1/1960	12/31/9999

S2 – ON <search_condition>

If specified, this part allows some conditions to be specified on the time-dependent component of each temporal composition of the resultant collection. A <search_condition> is formulated without any of our new temporal constructs in the same way as a conventional search condition in the OR SQL language [17] or in the core of SQL:2003, implemented in an ORDBMS. The evaluation S2 parallels the evaluation S1 with sequenced semantics. Therefore, we make an extension to the evaluation S1 with extra consideration on a <search_condition> to gain the evaluation S2 in Figure 8.7. Let tc_1, ..., tc_i, ..., tc_n be all temporal column references in a <value_expression> and/or a <search_condition> and their corresponding attribute histories be tc_1 = {($av_{1,1}$, $vt_{1,1}$), ..., ($av_{1,c1}$, $vt_{1,c1}$), ..., ($av_{1,h1}$, $vt_{1,h1}$)}, ..., tc_i = {($av_{i,1}$, $vt_{i,1}$), ..., ($av_{i,ci}$, $vt_{i,ci}$), ..., ($av_{i,hi}$, $vt_{i,hi}$)}, ..., and tc_n = {($av_{n,1}$, $vt_{n,1}$), ..., ($av_{n,cn}$, $vt_{n,cn}$), ..., ($av_{n,hn}$, $vt_{n,hn}$)}.

In addition to F, let P be a final function on these temporal column references like F. P only differs from F in respect of the data type of a returned value. A returned value of P is either TRUE or FALSE.

The evaluation S2 of a <value_expression> and a <search_condition> with sequenced semantics in parallel is shown to compute each temporal composition ($av_{j,c}$, $vt_{j,c}$) in the result of the evaluation S2.

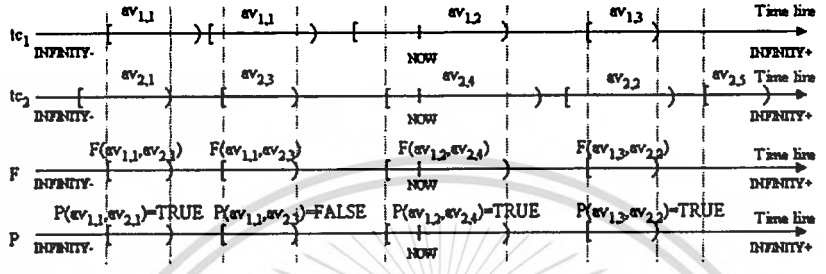


Figure 8.7 An Illustration on the Semantics Evaluation S2 of the ASELECT Temporal Construct

∀ c1 in 1..h1, ..., ci in 1..hi, ..., cn in 1..hn,
if $vt_{1,c1} \cap \ldots \cap vt_{i,ci} \cap \ldots \cap vt_{n,cn} \neq \emptyset$
and $F(av_{1,c1}, \ldots, av_{i,ci}, \ldots, av_{n,cn}) \neq$ NULL and $P(av_{1,c1}, \ldots, av_{i,ci}, \ldots, av_{n,cn}) =$ TRUE,
then for c in 1..h, $av_{j,c} = F(av_{1,c1}, \ldots, av_{i,ci}, \ldots, av_{n,cn})$; $vt_{j,c} = vt_{1,c1} \cap \ldots \cap vt_{i,ci} \cap \ldots \cap vt_{n,cn}$.

A sample result of the illustration in Figure 8.7 is obtained: ASELECT {F(tc_1, tc_2) OF <type_name> ON P(tc_1, tc_2)} = {(F($av_{1,1}$,$av_{2,1}$), $vt_{1,1} \cap vt_{2,1}$), (F($av_{1,2}$,$av_{2,4}$), $vt_{1,2} \cap vt_{2,4}$), (F($av_{1,3}$,$av_{2,2}$), $vt_{1,3} \cap vt_{2,2}$)}.

Example 8.19 - We can think of the history of each employee's name different from 'John'.

```
SELECT REFC AS ID, ASELECT {Name OF VARCHAR ON Name != 'John'} AS NAME
FROM Emp;
```

ID	NAME		
	vtValue	ValidTime	
		vtStart	vtEnd
E1	Ed	2/1/1982	1/1/1988
	Edward	1/1/1988	12/31/9999
E2	Di	1/1/1982	12/31/9999
E3	Johnson	1/1/1978	12/31/9999
E4	Jack	5/15/1950	12/31/9999
E5	White	12/1/1960	12/31/9999

S3 – <periods>

If <periods> is not specified, the resultant collection of temporal compositions {($av_{j,1}$, $vt_{j,1}$), ..., ($av_{j,c}$, $vt_{j,c}$), ..., ($av_{j,h}$, $vt_{j,h}$)} obtained from the evaluations S1 and S2 will be returned to temporal database users.

If <periods> is specified, this part allows a restriction on the time component of each temporal composition of the resultant collection obtained from the evaluations S1 and S2. After the evaluation S3, the resultant collection of temporal compositions is

returned at a derived temporal column to form a temporal table for temporal database users. With a confinement on validity, a resultant collection is {($av_{j,1}$, $vt_{j,1}$∩<periods>), ($av_{j,2}$, $vt_{j,2}$∩<periods>), ..., ($av_{j,c}$, $vt_{j,c}$∩<periods>), ..., ($av_{j,h}$, $vt_{j,h}$∩<periods>)}. If $vt_{j,c}$∩<periods> is an empty set, ($av_{j,c}$, $vt_{j,c}$∩<periods>) is excluded for any c in 1..h.

Example 8.20 - We can think of the history of something/someone in some particular periods of time, e.g. the history of each employee's name different from 'John' in 1985-1990.

```
SELECT REFC AS ID
     , ASELECT PERIOD('01-01-1985', '01-01-1991'){Name OF VARCHAR ON Name != 'John'} AS NAME
FROM Emp;
```

ID	NAME		
	vtValue	ValidTime	
		vtStart	vtEnd
E1	Ed	1/1/1985	1/1/1988
	Edward	1/1/1988	1/1/1991
E2	Di	1/1/1985	1/1/1991
E3	Johnson	1/1/1985	1/1/1991
E4	Jack	1/1/1985	1/1/1991
E5	White	1/1/1985	1/1/1991

8.3.3.4 GROUP BY Clause

Regarding temporal aggregations, [106] gives an overview on existing works that provide a support for aggregating temporal data in the tuple timestamping scheme. As for the attribute timestamping scheme, only [57, 60] support temporal aggregates. They allow the partitioning of a history along the time axis based on a temporal unit or duration. Aggregation is then made on these partitions. Differently, our work follows the spirit of the existing aggregation support of the OR SQL language [17] with the GROUP BY and/or HAVING clauses and aggregate functions in consideration on the temporal aspect of involved columns. An inherent support for non-procedurally aggregating temporal data is an essential part of our language. Indeed, [107] shows how complicated temporal data aggregation is with the non-temporal standard SQL language where the coalescing of resultant rows is explicitly done by database users.

If the GROUP BY and/or HAVING clauses are not specified, no grouping column is specified in the SELECT clause. The resultant table from the evaluation of the WHERE clause is considered to be only one group for aggregation if aggregate functions are used in the SELECT clause. Any existing aggregate function supplied by a DBMS is able to be applied to derived non-temporal columns and unable to derived temporal columns. So, part of a temporal extension for the support of the GROUP BY and/or HAVING clauses in our language is a collection of our temporal aggregate functions.

These functions are tCOUNT{}, tMAX{}, tMIN{}, tAVG{}, and tSUM{} where tCOUNT{} can be applied to temporal data of any data type, tMAX{} and tMIN{} can be applied to temporal data of a data type with a partial ordering such as alphanumeric and DATE data types, and the others only to temporal numeric data. Each of them takes one derived temporal column as an input. The formulation of a SELECT statement with aggregate functions is also described in the form of SELECT..FROM..WHERE shown in three previous subsections 8.3.3.1-8.3.3.3.

Example 8.21 – List the number of employees whose names are always different from 'John' and their highest salaries along time.

```
SELECT COUNT(REFC) AS EMP#, tMAX{ASELECT{Salary OF NUMERIC}} AS SAL_MAX
FROM Emp
WHERE ALWAYS {Name != 'John'};
```

EMP#	SAL_MAX		
	vtValue	ValidTime	
		vtStart	vtEnd
4	40	1/1/1980	1/1/1984
	50	1/1/1984	12/31/9999

If specified, the GROUP BY and HAVING clauses are shaped with conventional production rules in [17]. Like derived non-temporal columns, derived temporal columns might be present in the two clauses and aggregate functions. As we know, there are two parts of aggregation on an input table after the evaluation of the FROM and WHERE clauses: a set of so-called grouping columns and a set of aggregate function invocations in the SELECT clause. A set of grouping columns is a <grouping_element_list> and must be specified in the GROUP BY clause. A search condition in the HAVING clause is specified as the one in subsection 8.3.3.2 with or without aggregate function invocations and applied to each group from the partitioning of the rows of the input table returned by the evaluation of the GROUP BY clause. Using the specification of the GROUP BY and/or HAVING clauses, existing non-temporal aggregate functions, and our temporal aggregate functions, we show a non-procedural formulation of grouping and aggregation on a temporal table. The temporal semantics are examined below for three cases: grouping with temporal upward compatibility, grouping with no temporal aspect, and grouping with the temporal aspect.

i. Grouping with temporal upward compatibility

In this case, temporal column references might be present in the SELECT, GROUP BY, and/or HAVING clauses but their temporal aspect is not considered. None of our

new temporal constructs is allowed in the SELECT, GROUP BY, and/or HAVING clauses. Therefore, existing non-temporal aggregate functions are applied to temporal columns instead of our temporal aggregate functions. These temporal columns are treated as if they were non-temporal according to the principle of temporal upward compatibility.

Example 8.22 – List the highest salary at each department at present.

```
SELECT Dept AS DEPT_ID, MAX(Salary) AS SAL_MAX
FROM Emp
GROUP BY Dept;
```

DEPT_ID	SAL_MAX
D2	40
D1	50
D4	55
D3	50

ii. Grouping with no temporal aspect

Grouping with no temporal aspect is done with a <grouping_element_list> that contains one or many derived non-temporal columns and no derived temporal column. One or many non-temporal aggregate functions are independently invoked on derived non-temporal columns, and one or many temporal aggregate functions independently on derived temporal columns. The grouping of rows of the input table with no temporal aspect according to distinct values at the grouping derived non-temporal columns is evaluated in [17]. The evaluation of a non-temporal aggregate function invocation is carried out as normal. Thus, we pay attention to the evaluation of a temporal aggregate function invocation in this case. A temporal aggregate function invocation on a derived temporal column is accomplished by a user-defined aggregate function. At each iteration step, an input of this function is an attribute history at the derived temporal column of each row in the group determined by the grouping columns in a <grouping_element_list>. An attribute history comes out as an output of a temporal aggregate function at the termination stage of aggregation after the coalescing of temporal compositions of the attribute history currently achieved. Each temporal composition ($av_{j,c}$, $vt_{j,c}$) of the resultant attribute history shows aggregation $av_{j,c}$ of all values true for the same periods $vt_{j,c}$ of time. Temporal aggregation with a temporal aggregation function invocation in the SELECT clause is computed as follows for each group *g* of n rows determined by the grouping derived non-temporal columns.

Let r_1, ..., r_i, r_{i+1}, ..., r_n be n rows of the *g* group for n >= 1 and i in 1..n.

Let A_j be a derived temporal column on which a temporal aggregate function tAGG is invoked. tAGG can be tCOUNT{}, tMAX{}, tMIN{}, tAVG{}, or tSUM{}. At each iteration of aggregation, $A_j = \{(av_{i,1}, vt_{i,1}), \ldots, (av_{i,c}, vt_{i,c}), \ldots, (av_{i,h}, vt_{i,h})\}$, an attribute history of the derived temporal column in one r_i row.

For tCOUNT{} without the DISTINCT keyword, let $Ag = \{(av\_count_{j,1}, vt\_count_{j,1}), \ldots, (av\_count_{j,c\_count}, vt\_count_{j,c\_count}), \ldots, (av\_count_{j,h\_count}, vt\_count_{j,h\_count})\}$ be the previous outcome of aggregation at the previous stage. In case of the first stage, av_count_{j,c_count} and vt_count_{j,c_count} are initialization values.

For tMAX{}, $Ag=\{(av\_max_{j,1}, vt\_max_{j,1}), \ldots, (av\_max_{j,c\_max}, vt\_max_{j,c\_max}), \ldots, (av\_max_{j,h\_max}, vt\_max_{j,h\_max})\}$.

For tMIN{}, $Ag=\{(av\_min_{j,1}, vt\_min_{j,1}), \ldots, (av\_min_{j,c\_min}, vt\_min_{j,c\_min}), \ldots, (av\_min_{j,h\_min}, vt\_min_{j,h\_min})\}$.

For tAVG{}, $Ag\_sum=\{(av\_sum_{j,1}, vt_{j,1}), \ldots, (av\_sum_{j,c\_avg}, vt_{j,c\_avg}), \ldots, (av\_sum_{j,h\_avg}, vt_{j,h\_avg})\}$ and $Ag\_count =\{(av\_count_{j,1}, vt_{j,1}), \ldots, (av\_count_{j,c\_avg}, vt_{j,c\_avg}), \ldots, (av\_count_{j,h\_avg}, vt_{j,h\_avg})\}$.

For tSUM{}, $Ag=\{(av\_sum_{j,1}, vt\_sum_{j,1}), \ldots, (av\_sum_{j,c\_sum}, vt\_sum_{j,c\_sum}), \ldots, (av\_sum_{j,h\_sum}, vt\_sum_{j,h\_sum})\}$.

Step 1- Find a set *tpoint* of all distinct points in time present in all time components of all temporal compositions in both A_j and Ag: *tpoint* = $\{P_1, \ldots, P_k, P_{k+1}, \ldots, P_m\}$ where $P_1<\ldots<P_k<P_{k+1}<\ldots<P_m$.

Step 2 - Create a set *tinterval* of all intervals of time from *tpoint*: *tinterval* = $\{[P_1, P_2), \ldots, [P_k, P_{k+1}), \ldots, [P_{m-1}, P_m)\}$ where the end point of one interval will be the starting point of the next interval except for the last interval $[P_{m-1}, P_m)$. $|tinterval| = m-1$. Each interval $[P_k, P_{k+1})$ in *tinterval* will be associated with its appropriate initialization values up to the employed temporal aggregate function tAGG{}. In particular, for tCOUNT{}, $av\_count_k = 0$; for tMAX{}, av_max_k = a minimum numeric value; for tMIN{}, av_min_k = a maximum numeric value; for tAVG{}, $av\_sum_k = 0$ and $av\_count_k = 0$; for tSUM{}, $av\_sum_k = 0$.

Step 3 - Calculate an aggregation Ag for the current stage by examining the outcome of the previous stage and then the A_j input of the r_i row of the current stage.

For tCOUNT{}:
+ If vt_count_{j,c_count} contains $[P_k, P_{k+1})$, then $av\_count_k = av\_count_k + av\_count_{j,c\_count}$;
+ If $vt_{i,c}$ contains $[P_k, P_{k+1})$, then $av\_count_k = av\_count_k + 1$.
+ $Ag = \{\ldots, (av\_count_k, \{[P_k, P_{k+1})\}), \ldots\}$.

For tMAX{}:
+ If vt_max_{j,c_max} contains $[P_k, P_{k+1})$ and $av\_max_k < av\_max_{j,c\_max}$, then $av\_max_k = av\_max_{j,c\_max}$;
+ If $vt_{i,c}$ contains $[P_k, P_{k+1})$ and $av\_max_k < av_{i,c}$, then $av\_max_k = av_{i,c}$;
+ $Ag = \{\ldots, (av\_max_k, \{[P_k, P_{k+1})\}), \ldots\}$.

For tMIN{}:
+ If vt_min_{j,c_min} contains $[P_k, P_{k+1})$ and $av\_min_k > av\_min_{j,c\_min}$, then $av\_min_k = av\_min_{j,c\_min}$;
+ If $vt_{i,c}$ contains $[P_k, P_{k+1})$ and $av\_min_k > av_{i,c}$, then $av\_min_k = av_{i,c}$;
+ $Ag = \{\ldots, (av\_min_k, \{[P_k, P_{k+1})\}), \ldots\}$.

For tAVG{}:
+ If vt_{j,c_avg} contains $[P_k, P_{k+1})$, then $av\_sum_k = av\_sum_k + av\_sum_{j,c\_avg}$; $av\_count_k = av\_count_k + av\_count_{j,c\_avg}$;
+ If $vt_{i,c}$ contains $[P_k, P_{k+1})$, then $av\_sum_k = av\_sum_k + av_{i,c}$; $av\_count_k = av\_count_k + 1$;
+ $Ag\_sum = \{\ldots, (av\_sum_k, \{[P_k, P_{k+1})\}), \ldots\}$;
+ $Ag\_count = \{\ldots, (av\_count_k, \{[P_k, P_{k+1})\}), \ldots\}$.

For tSUM{}:
+ If vt_sum_{j,c_sum} contains $[P_k, P_{k+1})$, then $av\_sum_k = av\_sum_k + av\_sum_{j,c\_sum}$;
+ If $vt_{i,c}$ contains $[P_k, P_{k+1})$, then $av\_sum_k = av\_sum_k + av_{i,c}$;
+ $Ag = \{\ldots, (av\_sum_k, \{[P_k, P_{k+1})\}), \ldots\}$.

Step 4 - If the stage is a termination one, i.e. i=n, we compute an aggregation A*g* for tAVG{}: A*g* = {..., (av_sum_k/av_count_k, {[P_k, P_{k+1})}), ...}. After that, the coalescing on A*g* takes place as follows:

For all temporal compositions (av_k, vt_{k1}), (av_k, vt_{k2}), ..., (av_k, vt_{kc}) that have the same time dependent component av_k, a new temporal composition (av_k, $vt_{k1} \cup vt_{k2} \cup ... \cup vt_{kc}$) is created and added into A*g* while all coalesced temporal compositions (av_k, vt_{k1}), (av_k, vt_{k2}), ..., and (av_k, vt_{kc}) are removed from A*g*.

A*g* is returned as an output of a temporal aggregate function for the *g* group. It is also an attribute history where each time dependent component of a temporal composition is an aggregation value for some periods of time.

Example 8.23 – For each gender, list the number of employees and the highest salary that employees have been paid along time.

```
SELECT Gender, COUNT(REFC) AS EMP#, tMAX{ASELECT {Salary OF NUMERIC}} AS SAL_MAX
FROM Emp
GROUP BY Gender;
```

Gender	EMP#	SAL_MAX		
		vtValue	ValidTime	
			vtStart	vtEnd
M	3	40	1/1/1980	1/1/1984
		50	1/1/1984	1/1/1989
		55	1/1/1989	12/31/9999
F	2	30	1/1/1980	1/1/1984
		40	1/1/1984	9/1/1986
		50	9/1/1986	12/31/9999

iii. Grouping with the temporal aspect

In this case, grouping with the temporal aspect is done with a <grouping_element_list> that contains at least one derived temporal column and no derived non-temporal column. As of this moment, our language allows only one grouping column in the GROUP BY clause to be a derived temporal column. The rows of the input table are partitioned along time into groups. Each group is associated with a distinct value at the grouping derived temporal column and a period of time when that value is true in reality. In particular, let g_1, ..., g_k, ..., and g_m be m resulting groups of rows partitioned by the grouping column. For any k in 1..m, the g_k group is associated with (v_k, p_k) where v_k is a value which is the time-dependent component of some temporal composition at the derived temporal column and p_k is a period of time when v_k is valid in the modeled world. Given any two g_{k1} and g_{k2} groups for any k1 and k2 in 1..m and k1 <> k2, there is a constraint such that v_{k1} must be different from v_{k2} or if v_{k1} is equal to v_{k2} then p_{k1} must be disjoint from p_{k2}.

After the grouping along the time axis, non-temporal/temporal aggregation can be made on each g_k group of n rows of the input table with one or many non-temporal/temporal aggregate functions independently invoked on derived non-temporal/temporal columns, respectively. The evaluation of a non-temporal aggregate function invocation is obtained according to the SQL standard [17] while the one of a temporal aggregate function invocation has already been presented in the previous case 8.3.3.4.ii.

The evaluation steps of the GROUP BY clause with a grouping derived temporal column A_j with respect to valid time at the A_j column of the input temporal table *ut1* after the evaluation of the FROM and WHERE clauses are shown below. Let NA_1, NA_2, ..., and NA_{n1} be n1 derived non-temporal columns on which non-temporal aggregate functions $AGG_1()$, $AGG_2()$, ..., and $AGG_{n1}()$ are invoked, respectively. Let TA_1, TA_2, ..., and TA_{n2} be n2 derived temporal columns on which temporal aggregate functions $tAGG_1\{\}$, $tAGG_2\{\}$, ..., and $tAGG_{n2}\{\}$ are invoked, respectively. The query statement is now rewritten as follows with the specification of a derived temporal column in subsection 8.3.3.3. This statement is then transformed into a sequence of SQL statements that can be executed by an ORDBMS.

The transformation steps are presented as follows. Given the following statement:

```
SELECT ASELECT [<periods_Aj>] {Aj OF <type_name> [ON <search_condition>]},
        AGG1(NA1), ..., AGGn1(NAn1),
        tAGG1{ASELECT [<periods_TA1>] {TA1 OF <type_name> [ON <search_condition>]}}, ...,
        tAGGn2{ASELECT [<periods_TAn2>] {TAn2 OF <type_name> [ON <search_condition>]}}
FROM ut1
GROUP BY ASELECT [<periods>] {Aj OF <type_name> [ON <search_condition>]}
HAVING <search_condition>;
```

Step 1 – Associate each row with a value and a period of time at the grouping derived temporal column so that the timestamping of the A_j temporal column is now placed at the tuple level. In the following statement, tSelect() is our temporal function, which can be executed by the employed ORDBMS. It is used to evaluate the temporal construct ASELECT{}. The result of tSelect() is an attribute history. Also, the evaluation of ASELECT{} on TA_i for any i in 1..n2 restricts the valid time of the attribute history returned by $t.TA_i$ within the valid time of the attribute history returned by $t.A_j$.

```
CREATE VIEW ut2 AS
SELECT nt.vtValue AS Aj, vt.vtStart AS vtStart, vt.vtEnd AS vtEnd,
        NA1, ..., NAn1, tSelect(t.TA1) AS TA1, ..., tSelect(t.TAn2) AS TAn2
FROM ut1 t, TABLE (tSelect(t.Aj)) nt, TABLE (nt.ValidTime) vt;
```

Step 2 – Create a set *tpoint* of all distinct points in time at the grouping derived temporal column.

```
CREATE VIEW tpoint AS
SELECT vtStart AS Point FROM ut2 UNION SELECT vtEnd AS Point FROM ut2;
```

Step 3 – Create a set *tinterval* of all adjacent intervals of time.

```
CREATE VIEW tinterval AS
SELECT t1.Point AS vtStart, t2.Point AS vtEnd
FROM tpoint t1, tpoint t2
WHERE t1.Point < t2.Point AND NOT EXISTS (SELECT * FROM tpoint t3 WHERE t3.Point > t1.Point AND t3.Point
<t2.Point);
```

Step 4 – Adjust the interval of time associated with each row of *ut2* so that the intervals of time associated with any two different rows are either disjoint or equal.

```
CREATE VIEW ut3 AS
SELECT t.Aj AS Aj, vt.vtStart AS vtStart, vt.vtEnd AS vtEnd, NA1, ..., NAn1, TA1, ..., TAn2
FROM ut2 t, tinterval vt
WHERE t.vtStart <= vt.vtStart AND t.vtEnd >= vt.vtEnd;
```

Step 5 – Partition the rows of *ut3* and compute non-temporal/temporal aggregations with non-temporal/temporal aggregate functions, respectively.

```
CREATE VIEW taggregation AS
SELECT Aj, vtStart, vtEnd, AGG1(NA1) AS Ag1, ..., AGGn1(NAn1) AS Agn1, tAGG1(TA1) AS TAg1, ..., tAGGn2(TAn2) AS
TAgn2
FROM ut3
GROUP BY Aj, vtStart, vtEnd
HAVING <search_condition>;
```

Step 6 – Coalesce the result of the previous step to achieve a table *tresult* that includes two extra timestamping attributes vtStart and vtEnd. In the following statement, we employ the coalescing algorithm defined in [46]. Also, tEqual() is our predefined function to check the equality of two attribute histories. Two attribute histories are equal to each other if and only if identical to each other.

```
CREATE TABLE tresult AS
SELECT DISTINCT f.vtStart, l.vtEnd, f.Aj, f.Ag
FROM taggregation f, taggregation l
WHERE f.vtStart < l.vtEnd AND f.Aj = l.Aj AND f.Ag1 = l.Ag1 AND ... AND f.Agn1 = l.Agn1
AND tEqual(f.TAg1, l.TAg1) = 'TRUE' AND ... AND tEqual(f.TAgn2, l.TAgn2) = 'TRUE'
AND NOT EXISTS (SELECT * FROM taggregation m
                WHERE m.Aj = f.Aj AND m.Ag1 = f.Ag1 AND ... AND m.Agn1 = f.Agn1
                AND tEqual(m.TAg1, f.TAg1) = 'TRUE' AND ... AND tEqual(m.TAgn2, f.TAgn2) = 'TRUE'
                AND f.vtStart < m.vtStart AND m.vtStart < l.vtEnd
                AND NOT EXISTS (SELECT * FROM taggregation r1
                                WHERE r1.Aj = f.Aj AND r1.Ag = f.Ag
                                AND r1.vtStart < m.vtStart AND m.vtStart <= r1.vtEnd))
AND NOT EXISTS (SELECT * FROM taggregation r2
                WHERE r2.Aj = f.Aj AND r2.Ag1 = f.Ag1 AND ... AND r2.Agn1 = f.Agn1
                AND tEqual(r2.TAg1, f.TAg1) = 'TRUE' AND ... AND tEqual(r2.TAgn2, f.TAgn2) = 'TRUE'
                AND ((r2.vtStart < f.vtStart AND f.vtStart <= r2.vtEnd)
                OR (r2.vtStart <= l.vtEnd AND l.vtEnd < r2.vtEnd)));
```

Example 8.24 – List the number of employees whose names always start with 'J' at each department in 1980-1985.

```
SELECT ASELECT PERIOD('01-01-1980', '01-01-1986') {Dept OF REF(DEPT_T)} AS DEPT_ID,
       COUNT(REFC) AS EMP#
FROM Emp
```

WHERE ALWAYS {Name LIKE 'J%'}
GROUP BY ASELECT PERIOD('01-01-1980', '01-01-1986') {Dept OF REF(DEPT_T)};

DEPT_ID	vtStart	vtEnd	EMP#
D4	1/1/1980	1/1/1986	1
D3	1/1/1984	1/1/1986	1

Example 8.25 – List the highest salary at each department that has had less than 20 employees along time.

SELECT ASELECT {Dept OF REF(DEPT_T)} AS DEPT_ID, **tMAX{ASELECT {Salary OF NUMERIC}}** AS SAL_MAX
FROM Emp
GROUP BY ASELECT {Dept OF REF(DEPT_T)}
HAVING COUNT (REFC) < 20;

DEPT_ID	vtStart	vtEnd	SAL_MAX		
			vtValue	ValidTime	
				vtStart	vtEnd
D1	1/1/1982	12/31/9999	30	1/1/1982	8/1/1984
			40	8/1/1984	9/1/1986
			50	9/1/1986	12/31/9999
D2	4/1/1987	12/31/9999	40	4/1/1987	12/31/9999
D4	1/1/1980	1/1/1984	40	1/1/1980	1/1/1984
D4	1/1/1984	12/31/9999	45	1/1/1984	1/1/1989
			55	1/1/1989	12/31/9999
D3	1/1/1980	1/1/1984	30	1/1/1980	1/1/1984
D3	1/1/1984	12/31/9999	50	1/1/1984	12/31/9999

8.3.4 Temporal Data Modification

Temporal data modifications are made on temporal tables. These modifications include insert, delete, and update activities. Insertions are used for the recording of new tuples/objects into the database. Deletions are used for the removing of existing tuples/objects not true any longer from the database logically. Updates are used for the changing of some existing value at a column to another value true for some periods of time. Among existing SQL modification statements INSERT, DELETE, and UPDATE, only UPDATE statements perform at the attribute level while INSERT and DELETE statements at the tuple/object level.

All existing SQL modification statements work in our proposed language. Based on the principle of temporal upward compatibility, they will be interpreted as current modifications on temporal tables which are temporal modifications with the period [NOW, INFINITY+) of valid time.

Moreover, we introduce a new temporal construct VALIDTIME to all conventional SQL modification statements in order to include into each modification the periods of applicability provided by users. Also, we define a new temporal statement ADELETE which is a counterpart of the DELETE statement at the attribute level. The function of ADELETE is to remove the no longer valid information of some particular temporal elementary facts which are gathered together in the same row of a temporal table. It is needed because the normal DELETE statement performs at the tuple level and the

removal of the tuple impacts all facts grouped in the tuple. However, there is no need of such a new temporal construct for the existing UPDATE statements. It is because the update statements inherently perform at the attribute level. As for insertions, no new temporal construct at the attribute level is necessary because the recording of a new row of a temporal table is to insert all facts, both non-temporal and temporal, into the database. If the insertion of some extra information of a particular temporal fact is required, an update on that temporal fact is formed and a temporal update statement is issued instead.

For brevity, only an illustration on the specification of temporal data modifications is shown through some examples. The results as well as the consideration on whether or not any constraint is violated are ignored.

8.3.4.1 Insertion

The syntax of an INSERT statement in our language is defined as follows.

```
<tsql_insert_stmt> ::= INSERT INTO <temporal_table_name> <left_paren> <insert_column_list> <right_paren>
                       VALUES <left_paren> <insert_value_list> <right_paren>
<insert_value_list> ::= <value_expression> | <insert_value_list> <comma> <value_expression>
        | <temporal_value_expression> | <insert_value_list> <comma> <temporal_value_expression>
<temporal_value_expression> ::= <value_expression> VALIDTIME <periods>
        | <left_bracket> <value_expression> VALIDTIME <periods>
                         [{<comma> <value_expression> VALIDTIME <periods>}...] <right_bracket>
```

A temporal INSERT statement is different from a non-temporal INSERT statement since it includes one or many <temporal_value_expression>s. Each <temporal_value_expression> is in fact a value expression with the following of the VALIDTIME clause or a collection of combinations each of which comprises a value expression and the VALIDTIME clause. Each <temporal_value_expression> is then assigned to a temporal column of a new row of a temporal table. The meaning behind the VALIDTIME clause is to allow users to specify the periods of validity of each temporal elementary fact grouped in a new row of a temporal table. The specification and evaluation of a <value_expression> in the VALUES clause of an INSERT statement are determined as usual.

Example 8.26 - A new department has been established. The name is RD. The manager has not yet been assigned. The budget given approximately is 300. The information of the new department is now recorded using a non-temporal (current) insertion on the Dept temporal table.

```
INSERT INTO Dept (Department, Manager, Budget)
VALUES ('RD', NULL, 300);
```

Example 8.27 - Consider a temporal insertion on a temporal table. The statement is issued with the use of the new construct VALIDTIME to record the information of a new employee according to an employment plan with different periods of validity provided by the user. A new employee has a current name Jane. She was born on 01/21/1970. She hopefully works in the new department RD from 01/01/2010. Besides, her salary is known to be 35 in 1990-1994 and 40 from 1995.

```
INSERT INTO Emp (Name, Gender, D_birth, Salary, Dept)
VALUES ('Jane', 'F', TO_DATE('01-21-1970', 'mm-dd-yyyy'),
        {35 VALIDTIME PERIOD('01-01-1990', '01-01-1995'), 40 VALIDTIME PERIOD('01-01-1995', '12-31-9999')},
        (SELECT d.REFC FROM Dept d WHERE d.Department = 'RD') VALIDTIME PERIOD('01-01-2010', '12-31-
        9999'));
```

If the input table contains both LIFESPAN and temporal columns, the value at the LIFESPAN column of the new row (tuple) of the input table must be automatically and implicitly computed and inserted along with other values at other columns of the new row. For a non-temporal INSERT statement, i.e. a current INSERT statement, [NOW, INFINITY+) is used as the life span of the new row by default. For a temporal INSERT statement, the value at the LIFESPAN column of the new row is the union of the valid times of the values at all temporal columns A_{j1}, A_{j2}, A_{j3}, ... of the new row: LIFESPAN = A_{j1}.ValidTime $\cup$ A_{j1}.ValidTime $\cup$ A_{j1}.ValidTime $\cup$ ... These valid times are specified in the VALIDTIME clauses.

8.3.4.2 Deletion

Before the deletion operation performs, we need to determine which rows of a temporal table will be affected by the deletion operation. If the WHERE clause is specified, this determination is made by a <search_condition> which is defined in subsection 8.3.3.2. Otherwise, all rows present in a temporal table are taken into consideration. The syntax of a DELETE/ADELETE statement is described as follows.

```
<tsql_delete_stmt> ::= DELETE FROM <temporal_table_name> [WHERE <search_condition>]                (D1)
                     | DELETE FROM <temporal_table_name>                                           (D2)
                       VALIDTIME <periods> [WHERE <search_condition>]
                     | ADELETE FROM <temporal_table_name>                                          (D3)
                       ATTRIBUTES <left_paren> <temporal_column_list> <right_paren>
                       [WHERE <search_condition>]
<temporal_column_list> ::= <temporal_column_name> VALIDTIME <periods>
        | <temporal_column_list> <comma> <temporal_column_name> VALIDTIME <periods>
```

For the semantics explanation about production rules D1, D2, and D3, let *r* be one row in the resultant collection of rows after the evaluation of the WHERE clause if specified. Otherwise, let *r* be one row of the input temporal table. Let $r.A_j$ be a column reference that returns an attribute history at any A_j temporal column of the *r* row corresponding to a temporal elementary fact grouped in the *r* row. Using the definitions in subsection 3.3, $r.A_j = \{(av_{j,1}, vt_{j,1}), (av_{j,2}, vt_{j,2}), \ldots, (av_{j,c}, vt_{j,c}), \ldots, (av_{j,h}, vt_{j,h})\}$.

D1 – A DELETE statement formed by the production rule D1 will be evaluated at the tuple level according to the principle of temporal upward compatibility. No influence is made on non-temporal facts in the *r* row. The current information of all facts recorded in the *r* row is logically removed. Therefore, this deletion will adjust the validity of all temporal facts of a row to exclude the current period of time [NOW, INFINITY+) so that these facts can not be true for that period of time. Each temporal fact is achieved as: $r.A_j$ = {($av_{j,1}$, $vt_{j,1}$-{[NOW, INFINITY+)}), ..., ($av_{j,c}$, $vt_{j,c}$-{[NOW, INFINITY+)}), ..., ($av_{j,h}$, $vt_{j,h}$-{[NOW, INFINITY+)})}. If $vt_{j,c}$-{[NOW, INFINITY+)} is an empty set, $av_{j,c}$ is excluded for c in 1..h. If the LIFESPAN column exists, the value at the LIFESPAN column of the *r* row needs to be adjusted so that no implicit containment constraint is violated. This adjustment is implicitly made by the system after a successful deletion on the input table: *r*.LIFESPAN = *r*.LIFESPAN – {[NOW, INFINITY+)}. After that, if *r*.LIFESPAN is empty or NULL, the *r* row will be physically deleted from the input table.

Example 8.28 - Remove all data of the employees who currently work in the Sales department as the Sales department is going to be integrated into the Marketing department. This situation leads to a non-temporal deletion on the Emp temporal table at the tuple level.

```
DELETE FROM Emp
WHERE Dept.Department = 'Sales';
```

D2 – A DELETE statement formed by the production rule D2 will be evaluated in the same way as the one by D1 with the periods of time <periods> supplied by temporal database users instead of [NOW, INFINITY+). So, each temporal fact is achieved as: $r.A_j$ = {($av_{j,1}$, $vt_{j,1}$-<periods>), ..., ($av_{j,c}$, $vt_{j,c}$-<periods>), ..., ($av_{j,h}$, $vt_{j,h}$-<periods>)}. If $vt_{j,c}$-<periods> is an empty set, $av_{j,c}$ is excluded for c in 1..h. The consideration of the affect of a deletion on the LIFESPAN column if specified is similar to the previous case D1. The value at the LIFESPAN column of the *r* row needs to be adjusted to exclude the periods of time <periods> from the life span of the *r* row so that no implicit containment constraint is violated. This adjustment is implicitly made as: *r*.LIFESPAN = *r*.LIFESPAN – <periods>. After that, if *r*.LIFESPAN is empty or NULL, the *r* row will be physically deleted from the input table.

Example 8.29 – A temporal deletion at the tuple level: Remove all data of the employee whose name is Ed true for 1985-1986 because it is realized that he has been absent from his department for 1985-1986.

```
DELETE FROM Emp
VALIDTIME PERIOD('01-01-1985', '01-01-1987')
WHERE ALWAYS PERIOD('01-01-1985', '01-01-1987') {Name = 'Ed'};
```

D3 – The production rule D3 adapts the DELETE statement with a new temporal clause ADELETE at the attribute level to declare which temporal attributes particularly need to be considered for the removal of temporal elementary facts gathered in a single row. Each removal of a temporal elementary fact is also made for some user-supplied periods of time in a new temporal clause ATTRIBUTES. The evaluation of an ADELETE statement is the same as the one of D2 for specified temporal facts, not for all. So, $r.A_j$ = {($av_{j,1}$, $vt_{j,1}$-<periods>), ..., ($av_{j,c}$, $vt_{j,c}$-<periods>), ..., ($av_{j,h}$, $vt_{j,h}$-<periods>)} if A_j is specified in the ATTRIBUTES clause. If $vt_{j,c}$-<periods> is an empty set, $av_{j,c}$ is excluded for c in 1..h. The consideration on the LIFESPAN column is different from the two previous cases if the LIFEPSAN column is contained in the input table. The value at the LIFESPAN column of the *r* row has to be re-computed from the union of the valid times of the values at all temporal columns A_{j1}, A_{j2}, A_{j3}, ... of the *r* row: *r*.LIFESPAN = $r.A_{j1}$.ValidTime $\cup$ $r.A_{j1}$.ValidTime $\cup$ $r.A_{j1}$.ValidTime $\cup$ ... After that, if *r*.LIFESPAN is empty or NULL, the *r* row will be physically deleted from the input table.

Example 8.30 - Remove the salary data of the period 1988-1990 of all employees working at the Marketing department some time because it is realized that the computation of the salary has not been correct for that period. A temporal deletion on the Emp temporal table at the attribute level is issued.

```
ADELETE FROM Emp
ATTRIBUTES (Salary VALIDTIME PERIOD ('01-01-1988', '01-01-1991'))
WHERE ANYTIME (Dept.Department = 'Marketing');
```

8.3.4.3 Update

The syntax of an UPDATE statement in the proposed language is defined below:

```
<tsql_update_stmt> ::= UPDATE <temporal_table_name> SET <set_list> [WHERE <search_condition>]
<set_list> ::= <column_assignment> | <set_list> <comma> <column_assignment>
<column_assignment> := <column_name> <equals_operator> <value_expression>
        | <column_name> <equals_operator> <value_expression> VALIDTIME <periods>
```

For the semantics explanation, let *r* be one row in the resultant collection of rows after the evaluation of the WHERE clause if specified. A <search_condition> in the WHERE clause is defined in subsection 8.3.3.2. Otherwise, let *r* be one row of the input temporal table. Let A_j be a temporal column specified in the SET clause and $r.A_j$ be a column reference that returns an attribute history at any A_j temporal column of the *r* row corresponding to a temporal elementary fact grouped in the *r* row. From subsection 8.2, $r.A_j$ = {($av_{j,1}$, $vt_{j,1}$), ($av_{j,2}$, $vt_{j,2}$), ..., ($av_{j,c}$, $vt_{j,c}$), ..., ($av_{j,h}$, $vt_{j,h}$)}. Let av_{new} be a value expected to be assigned to A_j in the SET clause. The data type of av_{new} is AT_j compatible with the data type of A_j.

If A_j is specified in the SET clause without the VALIDTIME clause, the evaluation of $A_j = av_{new}$ follows the principle of temporal upward compatibility. The attribute history of a temporal fact corresponding to A_j is achieved as a collection of temporal compositions each of which is $(av_{j,c'}, vt_{j,c'})$:

- $\forall$ c in 1..h, if $av_{j,c} != av_{new}$ and $vt_{j,c} - \{[NOW, INFINITY+)\} != \emptyset$, then $av_{j,c'} = av_{j,c}$; $vt_{j,c'} = vt_{j,c} - \{[NOW, INFINITY+)\}$.
- $\forall$ c in 1..h, if $av_{j,c} != av_{new}$ and $vt_{j,c} - \{[NOW, INFINITY+)\} = \emptyset$, then $(av_{j,c}, vt_{j,c})$ is excluded.
- $\forall$ c in 1..h, if $av_{j,c} = av_{new}$, then $av_{j,c'} = av_{new}$; $vt_{j,c'} = vt_{j,c} \cup \{[NOW, INFINITY+)\}$. If there is no such $av_{j,c}$, then a new temporal composition $(av_{new}, \{[NOW, INFINITY+)\})$ is included.

Example 8.31 - Consider a following case of a non-temporal (current) update on the Emp temporal table. The employee whose current name is White has decided to move to the new department 'RD'. Therefore, her department is now 'RD'.

```
UPDATE Emp
SET Dept = (SELECT d.REFC FROM Dept d WHERE d.Department = 'RD')
WHERE Name = 'White';
```

If A_j is specified in the SET clause with the VALIDTIME clause, the evaluation of $A_j = av_{new}$ VALIDTIME <periods> is the same as the one without the VALIDTIME clause for the periods of time supplied by temporal database users instead of [NOW, INFINITY+). The attribute history of a temporal fact corresponding to A_j is achieved as a collection of temporal compositions each of which is $(av_{j,c'}, vt_{j,c'})$:

- $\forall$ c in 1..h, if $av_{j,c} != av_{new}$ and $vt_{j,c}$ - <periods> $\neq \emptyset$, then $av_{j,c'} = av_{j,c}$; $vt_{j,c'} = vt_{j,c}$ - <periods>.
- $\forall$ c in 1..h, if $av_{j,c} != av_{new}$ and $vt_{j,c}$ - <periods> $= \emptyset$, then $(av_{j,c}, vt_{j,c})$ is excluded.
- $\forall$ c in 1..h, if $av_{j,c} = av_{new}$, then $av_{j,c'} = av_{new}$; $vt_{j,c'} = vt_{j,c} \cup$ <periods>. If there is no such $av_{j,c}$, then a new temporal composition $(av_{new}$, <periods>) is included.

Example 8.32 - Consider a temporal update on the Salary temporal attribute of the Emp temporal table: the salary of an employee will be 60 for 1990-1995 if he/she works in the Sales department for 1990-1995.

```
UPDATE Emp
SET Salary = 60 VALIDTIME PERIOD ('01-01-1990', '01-01-1996')
WHERE ALWAYS PERIOD ('01-01-1990', '01-01-1996')
                {Dept = (SELECT REFC FROM Dept1 WHERE Department = 'Sales')};
```

Different from insertions and deletions, each update has to guarantee that no constraint on the input table is violated after the update. Therefore, only those updates satisfying all containment constraints defined on the input table are accepted. Otherwise, they are rejected. No modification is further made at the LIFESPAN column for update operations.

8.4 The Proposed Temporal Compatible Object Relational Database System

In this subsection, the architecture of the proposed temporal compatible object relational database system is presented in Figure 8.8.

To avoid the disadvantages of a layered implementation approach [53, 108, 109], the implementation of the proposed temporal compatible object relational SQL language is accomplished as an add-on that can be conveniently and directly integrated into an employed ORDBMS. Indeed, the language development is carried out using an ORDBMS for both embedded and interactive SQL modes. Using object relational technology, all existing interfaces of the employed ORDBMS can be inherited to offer non-temporal and temporal database users a temporal transparency environment. Our implementation is not product-specific to any ORDBMS. It can be portal to any ORDBMS that allows the extensibility of user-defined types, user-defined methods, and optimization methods. Such an ORDBMS can be Oracle 10g/11g with data cartridges, Informix with DataBlades, or Universal DB2 with Extenders.

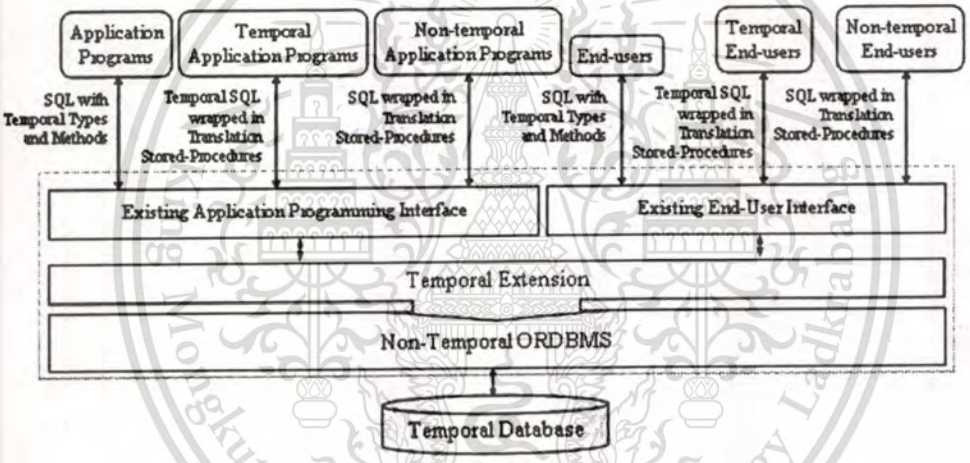


Figure 8.8 The Proposed Temporal Compatible Object Relational Database System

The temporal extension of the proposed temporal compatible object relational database system includes temporal abstract data types and temporal data methods that are used for the transformation of temporal SQL statements into conventional SQL statements temporally enhanced with our temporal data types and methods. Wrapped in our appropriate translation stored-procedure, a temporal SQL statement is directly submitted to the employed ORDBMS temporally enhanced with our temporal extension via an existing interface.

ALL conventional SQL statements with none of our temporal types and methods and none of our new temporal constructs are still supported on a temporal database in our system with the principle of temporal upward compatibility. They are issued by non-

temporal end-users and non-temporal application programs. To be able to be executed, these conventional SQL statements must be wrapped in our appropriate translation stored-procedures. With the inclusion of our new temporal constructs, SQL statements are now temporal SQL statements supported with the temporal semantics in the sequenced manner. These temporal SQL statements are formulated by temporal end-users and temporal application programs. They are also wrapped into our translation stored-procedures and then submitted to the employed ORDBMS for execution. More non-restrictively, end-users and application programs can use the proposed language in a non-sequenced manner by formulating SQL statements with our predefined temporal data types and methods and with none of the new temporal constructs introduced the previous subsection 8.3. These non-sequenced SQL statements are in fact conventional SQL statements; thus able to be directly submitted to the employed ORDBMS and able to be executed by the employed ORDBMS.

8.5 Summary

In this chapter, a novel temporal object relational SQL language has been developed with the valid time support. The main temporal support with the attribute timestamping scheme has been presented. Temporal data representation of the language is defined with the new concepts of temporal attributes, temporal structured types, temporal columns, and temporal tables. It is not only natural but also free from data redundancy and temporal data anomalies with the attribute timestamping scheme and the life span support by using object relational database technology. Indeed, the notion of history orientation can be reached. Moreover, our temporal support is generalized and independent of the time-varying of data of any particular data type. Supporting the temporal data representation, this temporal object relational database system can be used as an underlying system for temporal data warehouses such as the one proposed in [110].

In summary, the proposed language allows upward compatibility with the non-temporal object relational SQL language, temporal upward compatibility, temporal variants for sequenced cases, and non-restrictiveness for non-sequenced cases. It significantly facilitates temporal data management in a non-procedural manner and

serves a wide range of both non-temporal and temporal users. With the use of our language, many kinds of requirements can be asked against a single database with and without the awareness of its temporal aspect. By means of temporal operators ALWAYS, ANYTIME, NEXT, PREV, and UNTIL which are naturally linguistic from the temporal logic, users can deal with temporal data querying with ease and intuition. Furthermore, users are not required to study any new interface for the submission of temporal SQL statements to an ORDBMS with our integrated temporal extension, which is now part of the employed ORDBMS. This is because both non-temporal and temporal database users can work with our language in a temporal transparency environment via existing interfaces inherited from the employed ORDBMS in both interactive and embedded SQL modes. The language has been minutely examined with the sample temporal database adapted from the one in [101]. Although only attribute timestamping has been mainly done in our language, manipulations with tuple timestamping can be supported in a non-sequenced manner using our pre-defined temporal data types and methods.

In the future, the support of transaction time will be taken into account in order to achieve a fully temporal object relational SQL language with a new transaction time extension and/or by looking into the built-in transaction time support of an existing ORDBMS [68, 69]. It is also promising to include into our language a temporal extension of other SQL variants such as views, cursors, and assertions/triggers. As the first stage of our language development, performance issues of the proposed temporal compatible object relational database system are not within the scope of this work. Even though these features have not yet been supported, it is trivial to have them researched and included into the proposed work with no influence on the currently obtained temporal object relational database language and system through the object relational extensibility interface.

Chapter 9

A Temporal Object Relational Database for 3D Objects

As overviewed in chapter 5, a database for 3D objects is based on the object relational data model with the valid time support. That is, we aim at a temporal object relational database for 3D objects that are time-varying. There are two parts of such a temporal object relational database for 3D objects, namely the 3D graphical data part and the metadata part. The 3D graphical data part is generally handled for 3D objects of any application domain. It is the fixed time-invariant part of a temporal object relational database, called an object relational graphical database. In contrast, the metadata part, called a temporal object relational meta database, is particular to each application domain with regard to valid time. Therefore, the link between these two parts is established on the side of the metadata part. Moreover, the 3D graphical data part is considered at the logical level, i.e. at the implementation level so that the content of the 3D graphical data part directly comes from the 3D graphical data source via the graphical data population process. Unlike the 3D graphical data part, the metadata part is examined at the conceptual level, i.e. at the semantics level in order to capture more requirements related to each individual application domain for additional non-temporal/temporal non-explicit metadata and integrity constraints. The subsections 9.1 and 9.2 will detail these two 3D graphical data and metadata parts, respectively.

9.1 The 3D Graphical Data Part

Using object relational database technology, we consider the input 3D graphical data source at the graphical data level, i.e. in terms of graphical elements, for the 3D graphical data part.

As assumed in chapter 1, the handling of the 3D graphical data part of 3D objects and their scenes is done with the VRML language and the scene creation mechanism by laying out 3D objects in the scene. These 3D objects are generated from 3D patterns available in a previously established VRML graphical library. It is firstly argued that the employment of the VRML language does not narrow the proposed work down. It is due to the popularity of the VRML language. Also, the VRML language is an open standard

for describing 3D objects and their scenes. It is widely supported with the import and export utilities by many 3D authoring tools and toolkits. Secondly, the creation of a scene from existing 3D patterns (3D models) is expected to be acceptable because the reuse of 3D patterns worldwide over the Internet is practical. Thirdly, it is appropriately accepted that the scene creation mechanism is based on the bottom-up 3D graphics design approach with the widely-used divide-and-conquer strategy.

According to the description of a 3D scene graph shown Figure 5.2, a 3D scene graph contains one or many nodes and one node contains one or many fields. A node is an instance of a node type and a field is based on a field type. Only graphical nodes of the 3D scene graph are examined for visualization. With the use of the VRML language, these graphical nodes are geometry nodes for describing the form and orientation of a 3D object, visible nodes including Shape nodes for describing simple non-composite objects and grouping nodes for collecting descriptions or building complex composite objects, and attribute nodes for describing optical properties such as the position, color, and texture of a 3D object.

- Geometry node types: Box, Cone, Cylinder, ElevationGrid, Extrusion, IndexedLineSet, IndexedFaceSet, PointSet, Sphere, and Text;
- Visible node types: Shape and grouping node types including Transform, Group, Switch, and LOD;
- Attribute node types: Appearance, Material, ImageTexture, MovieTexture, PixelTexture, TextureTransform, Color, Coordinate, TextureCoordinate, Normal, and FontStyle.

Apart from the graphical node types, a collection of field types whose instances are values for the fields of a VRML node are SFBool, SFColor, MFColor, SFFloat, MFFloat, SFImage, SFInt32, MFInt32, SFNode, MFNode, SFRotation, MFRotation, SFString, MFString, SFTime, MFTime, SFVec2f, MFVec2f, SFVec3f, and MFVec3f. The SF prefix is for single valued field types and the MF prefix for collection (multi-valued) field types. Among these field types, the field types SFNode and MFNode are of importance to the relationship establishment among nodes in a 3D scene graph. Some relationships among all node types listed above are given in Figure 9.1 according to the specification of the VRML language.

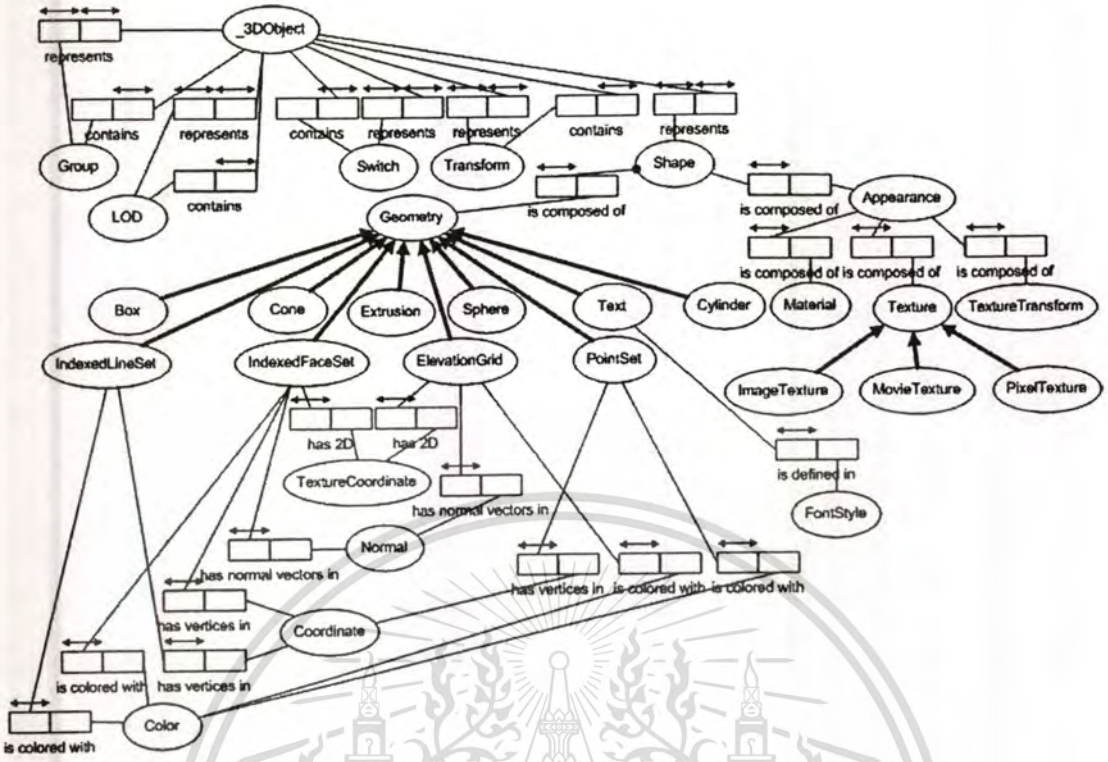


Figure 9.1 A NIAM Conceptual Schema Representing VRML Graphical Elements

In order to populate the entire content of a 3D graphical data source into the object relational graphical database, the information of all nodes of a 3D scene graph and the information of all fields of each node of the 3D scene graph need to be extracted and stored in such a way that there is no information loss. In other words, the whole of a 3D scene graph can be reconstructed from the graphical information kept in the object relational graphical database after the graphical data population process. So, by means of object relational database constructs, consideration on each field type is first determined, then consideration on each node type is defined, and finally the connections among nodes in the input 3D scene graph are established.

9.1.1 The Field Type Mapping

Table 9.1 shows the mapping of the field types to corresponding data types so that all permitted values for a field of a node in the VRML language can be recognized by the proposed temporal database system and properly stored in the object relational graphical database. The field types SFBool, SFFloat, SFInt32, SFString, and SFTime are mapped to built-in data types VARCHAR2, FLOAT, INTEGER, VARCHAR2, and

BINARY_DOUBLE, respectively. The others are mapped to user-defined types where the multi-valued field types with the MF prefix to collection types.

Table 9.1 The Mapping of the Field Types

Field Type	Corresponding Data Type	Corresponding Data Type Specification at the Implementation Level
SFBool	VARCHAR2	VARCHAR2(5)
SFColor	float3_t	CREATE TYPE float3_t AS ARRAY(3) OF FLOAT
MFColor	m_float3_t	CREATE TYPE m_float3_t AS TABLE OF float3_t
SFFloat	FLOAT	FLOAT
MFFloat	m_float_t	CREATE TYPE m_float_t AS TABLE OF FLOAT
SFImage	m_float_t	CREATE TYPE m_float_t AS TABLE OF FLOAT
SFInt32	INTEGER	INTEGER
MFInt32	m_integer_t	CREATE TYPE m_integer_t AS TABLE OF INTEGER
SFNode	node_t	CREATE TYPE node_t (NodeType NUMBER , NodeID SYS.ANYDATA)
MFNode	m_node_t	CREATE TYPE m_node_t AS TABLE OF node_t
SFRotation	float4_t	CREATE TYPE float4_t AS ARRAY(4) OF FLOAT
MFRotation	m_float4_t	CREATE TYPE m_float4_t AS TABLE OF float4_t
SFString	VARCHAR2	VARCHAR2(1000)
MFString	m_varchar2_t	CREATE TYPE m_varchar2_t AS TABLE OF VARCHAR2(1000)
SFTime	BINARY_DOUBLE	BINARY_DOUBLE
MFTime	m_double_t	CREATE TYPE m_double_t AS TABLE OF BINARY_DOUBLE
SFVec2f	float2_t	CREATE TYPE float2_t AS ARRAY(2) OF FLOAT
MFVec2f	m_float2_t	CREATE TYPE m_float2_t AS TABLE OF float2_t
SFVec3f	float3_t	CREATE TYPE float3_t AS ARRAY(3) OF FLOAT
MFVec3f	m_float3_t	CREATE TYPE m_float3_t AS TABLE OF float3_t

Among these field types, we pay attention to the field type SFNode corresponding to the data type node_t. Each value of the field type SFNode forms a reference of a node to another one, i.e., an arc of a 3D scene graph. A referred node can be a node of any node type: Shape, Transform, Group, LOD, or Switch. So, there are two attributes of the data type node_t: NodeType and NodeID. In Table 9.2, if NodeType is 0, a referred node is of the Shape node type; 1, Transform; 2, Group; 3, LOD; and 4, Switch.

Table 9.2 The Corresponding of a Value of the NodeType Attribute and the Node Type of a Referred Node

NodeType	The Node Type of a Corresponding Referred Node
0	Shape
1	Transform
2	Group
3	LOD
4	Switch

9.1.2 The Node Type Mapping

Different from the field types, each node type is uniformly mapped to a structured type as summarized in Table 9.3. Each field of the node type corresponds to an attribute of the structured type. The data type of the attribute is the one mapped from the field type of its corresponding field in Table 9.1.

Table 9.3 The Mapping of the Node Types

Node Types	Corresponding Structured Types
- Geometry: Box, Cone, Cylinder, ElevationGrid, Extrusion, IndexedLineSet, IndexedFaceSet, PointSet, Sphere, and Text	Box, Cone, Cylinder, ElevationGrid, Extrusion, IndexedLineSet, IndexedFaceSet, PointSet, Sphere, and Text
- Visible: Shape, Transform, Group, Switch, and LOD	Shape, Transform, Group_t, Switch, and LOD
- Attribute: Appearance, Material, ImageTexture, MovieTexture, PixelTexture, TextureTransform, Color, Coordinate, TextureCoordinate, Normal, and FontStyle	Appearance_t, Material_t, ImageTexture, MovieTexture, PixelTexture, TextureTransform_t, Color_t, Coordinate, TextureCoordinate, Normal_t, and FontStyle_t

Among these node types, the Shape, Transform, Group, LOD, and Switch node types are of interest as able to graphically represent some 3D object of interest in UoD. The definitions of the structured types stemming from these node types are given in Figure 9.2.

```
CREATE TYPE ShapeAS (
        appearance          Appearance_t            --              NULL
        , geometry          SYS.ANYDATA             --              NULL
        , MEMBER FUNCTION toVRML RETURN VARCHAR2
        , MEMBER FUNCTION compBBox RETURN MDSYS.SDO_GEOMETRY ... );
CREATE TYPE Transform AS (
        center                  float3_t        -- SFVec3f      0 0 0
        , children              m_node_t        -- MFNode
        , rotation              float4_t        -- SFRotation   0 0 1 0
        , scale                 float3_t        -- SFVec3f      1 1 1
        , scaleOrientation      float4_t        -- SFRotation   0 0 1 0
        , translation           float3_t        -- SFVec3f      0 0 0
        , bboxCenter            float3_t        -- SFVec3f      0 0 0
        , bboxSize              float3_t        -- SFVec3f      -1 -1 -1
        , MEMBER FUNCTION toVRML RETURN VARCHAR2
        , MEMBER FUNCTION compBBox RETURN MDSYS.SDO_GEOMETRY ... );
CREATE TYPE Group_t AS (
        children            m_node_t        -- MFNode
        , bboxCenter        float3_t        -- SFVec3f      0 0 0
        , bboxSize          float3_t        -- SFVec3f      -1 -1 -1
        , MEMBER FUNCTION toVRML RETURN VARCHAR2
        , MEMBER FUNCTION compBBox RETURN MDSYS.SDO_GEOMETRY ... );
CREATE TYPE LOD AS (
        alevel              m_node_t        -- MFNode
        , center            float3_t        -- SFVec3f      0 0 0
        , range             m_float_t       -- MFFloat
        , MEMBER FUNCTION toVRML RETURN VARCHAR2
        , MEMBER FUNCTION compBBox RETURN MDSYS.SDO_GEOMETRY ... );
CREATE TYPE Switch AS (
        choice              m_node_t        -- MFNode       []
        , whichChoice       INTEGER         -- SFInt32      -1
        , MEMBER FUNCTION toVRML RETURN VARCHAR2
        , MEMBER FUNCTION compBBox RETURN MDSYS.SDO_GEOMETRY ... );
```

Figure 9.2 The Definitions of the Structured Types Shape, Transform, Group_t, LOD, and Switch Stemming from the Shape, Transform, Group, LOD, and Switch Node Types, respectively

In the definitions of these structured types, there are two important member functions, toVRML() and compBBox(). The toVRML() member function is used to generate the VRML graphical data description of a 3D object graphically represented by a node GO of the node type corresponding to its structured type where the node GO corresponds to the instance of the structured type on which the function performs. The

compBBox() member function is used to compute the minimum bounding box of a 3D object graphically represented by a node GO of the node type corresponding to its structured type. It is bewared that the bboxCenter and bboxSize fields of the Transform and Group node types are also used to determine the bounding box of a 3D object graphically represented by a node of these two types. However, the values of these two fields must be explicitly specified. In practice, they are seldom given. Therefore, the inclusion of the compBBox() member function is necessary. The result of the compBBox() member function is specified to be of some existing 3D spatial data type supplied by the employed ORDBMS. In the implementation of this work, the SDO_GEOMETRY spatial data type of Oracle Spatial [111] is used with the Cartesian, right-handed, three-dimensional coordinate system. More operations on instances of these structured types corresponding to nodes of the mapped node types can be supplemented later by means of user-defined methods.

For the maintenance of the transformation hierarchical structure of a 3D scene and the composition relationships among 3D objects in the scene, we examine the node_t and m_node_t data types corresponding to the MFNode and SFNode field types. In the previous subsection, a permitted value of the NodeType attribute of the node_t data type corresponding to the SFNode field type has been discussed. Up to the value assigned to the NodeType attribute, the value assigned to the NodeID attribute will be a reference to an instance of one of the structured types Shape, Transform, Group_t, LOD, and Switch. That instance corresponds to a node that might graphically represent some 3D object. A link between the instance containing the reference and that instance has been established by means of the value assigned to the NodeID attribute. Such a link can be a composition relationship or a spatial relationship in a transformation hierarchy. Thus, the hierarchical structure of an entire scene can be preserved with our mappings of the field types and node types whereas each node of the scene is an instance of a structured type corresponding to the node type of that node and each field of that node is a value whose data type is a data type corresponding to the field type of that field.

9.1.3 The Object Relational Graphical Database for 3D Objects

Among nodes, only nodes of the node types Shape, Transform, Group, LOD, and Switch might graphically represent some 3D object or some part of a 3D object. Thus, we capture the graphical information of 3D objects in a 3D graphical data source at the node level. The object relational graphical database for 3D objects is defined as a collection of typed tables Shape_tab, Transform_tab, Group_tab, Lod_tab, and Switch_tab defined on the structured types Shape, Transform, Group_t, LOD, and Switch corresponding to the node types Shape, Transform, Group, LOD, and Switch. These typed tables are called graphical typed tables shown in Figure 9.3.

```
CREATE TABLE Shape_tab OF Shape;
CREATE TABLE Transform_tab OF Transform
NESTED TABLE children STORE AS t_children_nt;
CREATE TABLE Group_tab OF Group_t
NESTED TABLE children STORE AS g_children_nt;
CREATE TABLE Lod_tab OF LOD
NESTED TABLE alevel STORE AS l_children_nt
NESTED TABLE range STORE AS l_range_nt;
CREATE TABLE Switch_tab OF Switch
NESTED TABLE choice STORE AS s_children_nt;
```

Figure 9.3 Graphical Typed Tables Shape_tab, Transform_tab, Group_tab, Lod_tab, and Switch_tab of the Object Relational Graphical Database for 3D Objects

Each tuple/row of these graphical typed tables corresponds to a node in the scene graph. Relationships among tuples/rows of these graphical typed tables correspond to the relationships among nodes in the scene graph, i.e. the arcs of the scene graph. These relationships are maintained via the values of an attribute of each of the structured types Shape, Transform, Group_t, LOD, and Switch whose data type is the m_node_t data type. For no data redundancy, no duplicate is allowed in these graphical typed tables. As a result, two nodes of the same node type that have the same description will be associated with one single tuple/row of a graphical typed table defined on the structured type corresponding to that common node type.

With the availability of the object relational graphical database, any part of a scene can be re-formed from the graphical descriptions of appropriately selected rows/tuples that correspond to nodes of the expected part. Those graphical descriptions are obtained as the outputs of the invocations of the toVRML() member function. Nevertheless, doing post-processing might be needed to make the result interpretable to a target 3D graphics browser.

As for the case study, the object relational graphical database is presented with Tables 9.4 and 9.5. The graphical data in these two graphical typed tables is of 3D patterns in the graphics library and of 3D objects in the 3D graphical data sources.

Table 9.4 The Shape_tab Graphical Typed Table

REFC	appearance	geometry
shp1	Appearance_t (Material_t (1, float3_t (0, 0, 0.823529), float3_t (0, 0, 0), 0.2, float3_t (0, 0, 0), 0), NULL, NULL)	SYS.ANYDATA.convertObject (IndexedFaceSet (NULL, Coordinate (m_float3_t (float3_t (12.5, 0, 12.5), float3_t (12.5, 0, -12.5), float3_t (-12.5, 0, 12.5), float3_t (-12.5, 0, -12.5))), NULL, NULL, 'TRUE', m_integer_t(), 'FALSE', 'FALSE', m_integer_t (2, 0, 1, 3, -1), 0, m_integer_t(), 'TRUE', 'TRUE', m_integer_t()))
shp2	Appearance_t (Material_t (1, float3_t (0, 0.6, 0.423529), float3_t (0, 0, 0), 0.2, float3_t (0, 0, 0), 0), NULL, NULL)	SYS.ANYDATA.convertObject (IndexedFaceSet (NULL, Coordinate (m_float3_t (float3_t (12.5, 0, 12.5), float3_t (12.5, 0, -12.5), float3_t (-12.5, 0, 12.5), float3_t (-12.5, 0, -12.5))), NULL, NULL, 'TRUE', m_integer_t(), 'FALSE', 'FALSE', m_integer_t (2, 0, 1, 3, -1), 0, m_integer_t(), 'TRUE', 'TRUE', m_integer_t()))
shp3	Appearance_t (Material_t (1, float3_t (0.32549, 0.331373, 0.2313726), float3_t (0, 0, 0), 0.2, float3_t (0, 0, 0), 0), NULL, NULL)	SYS.ANYDATA.convertObject (Box (float3_t (8, 20, 10)))
shp4	Appearance_t (Material_t (1, float3_t (0.52549, 0.431373, 0.0313726), float3_t (0, 0, 0), 0.2, float3_t (0, 0, 0), 0), NULL, NULL)	SYS.ANYDATA.convertObject (Box (float3_t (8, 20, 10)))
shp5	Appearance_t (Material_t (1, float3_t (0.52549, 0.431373, 0.0313726), float3_t (0, 0, 0), 0.2, float3_t (0, 0, 0), 0), NULL, NULL)	SYS.ANYDATA.convertObject (Box (float3_t (6, 20, 4)))
shp6	Appearance_t (Material_t (1, float3_t (0.52549, 0.431373, 0.0313726), float3_t (0, 0, 0), 0.2, float3_t (0, 0, 0), 0), NULL, NULL)	SYS.ANYDATA.convertObject (Box (float3_t (6, 20, 6)))
shp7	Appearance_t (Material_t (1, float3_t (0.301961, 0.501961, 0.823529), float3_t (0, 0, 0), 0.2, float3_t (0, 0, 0), 0), NULL, NULL)	SYS.ANYDATA.convertObject (IndexedFaceSet (NULL, Coordinate (m_float3_t (float3_t (8.91304, 0, 1.91304), float3_t (8.91304, 0, -1.91304), float3_t (-8.91304, 0, 1.91304), float3_t (-8.91304, 0, -1.91304))), NULL, NULL, 'TRUE', m_integer_t(), 'FALSE', 'FALSE', m_integer_t (2, 0, 1, 3, -1), 0, m_integer_t(), 'TRUE', 'TRUE', m_integer_t()))
shp8	Appearance_t (Material_t (1, float3_t (0.301961, 0.501961, 0.823529), float3_t (0, 0, 0), 0.2, float3_t (0, 0, 0), 0), NULL, NULL)	SYS.ANYDATA.convertObject (IndexedFaceSet (NULL, Coordinate (m_float3_t (float3_t (1.91304, 0, 9.3913), float3_t (1.91304, 0, -9.3913), float3_t (-1.91304, 0, 9.3913), float3_t (-1.91304, 0, -9.3913))), NULL, NULL, 'TRUE', m_integer_t(), 'FALSE', 'FALSE', m_integer_t (2, 0, 1, 3, -1), 0, m_integer_t(), 'TRUE', 'TRUE', m_integer_t()))

Table 9.5 The Transform_tab Graphical Typed Table

REFC	center	children	rotation
tf1	float3_t(0, 0, 0)	m_node_t (node_t(0, SYS.ANYDATA.convertUROWID (shp4)))	float4_t(0, 0, 1, 0)
tf2	float3_t(0, 0, 0)	m_node_t (node_t(0, SYS.ANYDATA.convertUROWID (shp5)))	float4_t(0, 0, 1, 0)
tf3	float3_t(0, 0, 0)	m_node_t (node_t(1, SYS.ANYDATA.convertUROWID(tf1)), node_t(1, SYS.ANYDATA.convertUROWID(tf2)))	float4_t(0, 0, 1, 0)
tf4	float3_t(0, 0, 0)	m_node_t (node_t(0, SYS.ANYDATA.convertUROWID (shp6)))	float4_t(0, 0, 1, 0)
tf5	float3_t(0, 0, 0)	m_node_t (node_t(1, SYS.ANYDATA.convertUROWID(tf1)), node_t(1, SYS.ANYDATA.convertUROWID(tf2)), node_t(1, SYS.ANYDATA.convertUROWID(tf4)))	float4_t(0, 0, 1, 0)
tf6	float3_t(0, 0, 0)	m_node_t (node_t(0, SYS.ANYDATA.convertUROWID (shp1)))	float4_t(0, 0, 1, 0)
tf7	float3_t(0, 0, 0)	m_children_t (child_t(0, SYS.ANYDATA.convertUROWID (shp1)))	float4_t(0, 0, 1, 0)
tf8	float3_t(0, 0, 0)	, m_children_t (child_t(0, SYS.ANYDATA.convertUROWID (shp2)))	float4_t(0, 0, 1, 0)
tf9	float3_t(0, 0, 0)	m_children_t (child_t(0, SYS.ANYDATA.convertUROWID (shp1)))	float4_t(0, 0, 1, 0)
tf10	float3_t(0, 0, 0)	m_children_t (child_t(1, SYS.ANYDATA.convertUROWID(tf1)), child_t(1, SYS.ANYDATA.convertUROWID(tf2)))	float4_t(0, -1, 0, 1.5708)
tf11	float3_t(0, 0, 0)	m_children_t (child_t(0, SYS.ANYDATA.convertUROWID(shp3)))	float4_t(0, 0, 1, 0)
tf12	float3_t(0, 0, 0)	m_children_t (child_t(0, SYS.ANYDATA.convertUROWID (shp2)))	float4_t(0, 0, 1, 0)
tf13	float3_t(0, 0, 0)	m_children_t (child_t(0, SYS.ANYDATA.convertUROWID (shp7)))	float4_t(0, 0, 1, 0)
tf14	float3_t(0, 0, 0)	m_children_t (child_t(0, SYS.ANYDATA.convertUROWID (shp1)))	float4_t(0, 0, 1, 0)
tf15	float3_t(0, 0, 0)	m_children_t (child_t(0, SYS.ANYDATA.convertUROWID (shp2)))	float4_t(0, 0, 1, 0)
tf16	float3_t(0, 0, 0)	m_children_t (child_t(0, SYS.ANYDATA.convertUROWID (shp3)))	float4_t(0, 0, 1, 0)

Table 9.5 (cont.)

REFC	scale	scaleOrientation	translation	bboxCenter	bboxSize
tf1	float3_t(1, 1, 1)	float4_t(0, 0, 1, 0)	float3_t(0, 0, 0)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf2	float3_t(1, 1, 1)	float4_t(0, 0, 1, 0)	float3_t(7.17391, 0, -2.86957)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf3	float3_t(1, 1, 1)	float4_t(0, 0, 1, 0)	float3_t(-3.08696, 10, 0)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf4	float3_t(1, 1, 1)	float4_t(0, 0, 1, 0)	float3_t(0, 0, -7.65217)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf5	float3_t(1, 1, 1)	float4_t(0, 0, 1, 0)	float3_t(-3.08696, 10, 0)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf6	float3_t(1.23331, 1, 1.46663)	float4_t(0, 0, 1, 0)	float3_t(27.5264, 0, 49.9535)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf7	float3_t(1.4433, 1, 1.46663)	float4_t(0, 0, 1, 0)	float3_t(60.482, 0, 49.9535)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf8	float3_t(1.02333, 1, 1.46663)	float4_t(0, 0, 1, 0)	float3_t(90.8127, 0, 49.9535)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf9	float3_t(3.65978, 1, 1)	float4_t(0, 0, 1, 0)	float3_t(57.8572, 0, 80.2843)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf10	float3_t(1.41152, 0.825015, 1)	float4_t(0, 1, 0, 3.14159)	float3_t(25.1933, 8.25015, 43.2631)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf11	float3_t(1.58328, 0.650029, 1.8166)	float4_t(0, 0, 1, 0)	float3_t(64.8566, 6.50029, 47.0371)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf12	float3_t(2.44654, 1, 1.46663)	float4_t(0, 0, 1, 0)	float3_t(73.0226, 0, 49.9535)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf13	float3_t(5.12282, 1, 1)	float4_t(0, 0, 1, 0)	float3_t(57.8572, 0, 30.1219)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf14	float3_t(3.84643, 1, 1)	float4_t(0, 0, 1, 0)	float3_t(60.1903, 0, 80.2843)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf15	float3_t(2.6332, 1, 1.46663)	float4_t(0, 0, 1, 0)	float3_t(75.3557, 0, 49.9535)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)
tf16	float3_t(3.69769, 0.504208, 1)	float4_t(0, 0, 1, 0)	float3_t(47.0664, 5.04208, 81.4509)	float3_t(0, 0, 0)	float3_t(-1, -1, -1)

9.2 The Metadata Part

9.2.1 An Overview on the Metadata Part

Different from the 3D graphical data part, the metadata part is domain-specific. In order to capture more requirements not related to the content of an input 3D graphical data source, the metadata part should be examined at the conceptual level, i.e. at the semantics level before at the logical level, i.e. at the implementation level. As presented in chapter 7, a TOONIAM conceptual meta schema is defined from an input 3D graphical data source to represent metadata about 3D objects with respect to valid time at the abstract level. In this subsection, we concentrate on the metadata part at the implementation level based on the temporal object relational data model detailed in subsection 8.2. The temporal object relational meta database schema for the metadata part of temporal 3D objects is systematically and automatically obtained by the generally proposed conceptual/logical transformation procedure of a TOONIAM conceptual schema. Non-tempora/temporal typed tables are then defined on the achieved non-temporal/temporal structured types of the temporal object relational meta database schema as meta tables about temporal 3D objects, their scenes, and their categories.

9.2.2 The Conceptual/Logical Transformation Procedure of a Temporal Object Oriented NIAM Conceptual Schema

This subsection presents an automatic algorithm to transform a TOONIAM conceptual schema into a temporal object relational database schema using the notions of temporal structured types and temporal typed tables defined in subsection 8.2. This support increases the applicability of the proposed TOONIAM conceptual schema model to temporal object oriented data modeling in general. In the dissertation work, this conceptual/logical transformation procedure is applied on a TOONIAM conceptual meta schema so that a temporal object relational meta database schema can be systematically and automatically reached for temporal 3D objects.

The proposed conceptual/logical transformation procedure is a bottom-up process as the transformation of sub schemas is made before the one of the main schema. The logical data model employed in the transformation procedure is the object relational data model [17] with the valid time support that has been defined in chapter 8. As earlier introduced, the resultant database schema is a collection of structured types with and without temporal aspects. Typed tables are then defined on those structured types. Using the notions of non-temporal/temporal structured types and typed tables, the transformation procedure is carried out step by step as follows.

9.2.2.1 The Transformation of Sub Schemas

Let OT* be the object type which is further defined by the sub schema being considered.

Step I.1. *Consider the temporal aspect of OT*.*

If OT* is temporal and no structured type has been associated with OT*, a temporal structured type OT*_t with the LIFE SPAN requirement is created for OT*. Otherwise, a structured type OT*_t without the LIFE SPAN requirement is created for OT*.

Step I.2. *Consider each reference type in which OT* plays a role.*

The label type LT of the reference type is mapped to an attribute of the structured type OT*_t. If the role of LT is not required to be unique in a one-to-many reference type or the reference type is many-to-many, the data type of the attribute is a collection type (ARRAY or MULTISET) based on the data type DT specified for the label type LT.

Otherwise, the data type of the attribute is DT. If the reference type is temporal, then the attribute is temporal with some user-specified granularity or DAY by default.

Step I.3. *Consider each unary fact type with the role name UFT with which OT* plays.*

The fact type is mapped to an attribute of the structured type OT*_t. The data type of the attribute is a binary valued type such as BOOLEAN. If the fact type is temporal, then the attribute is temporal with some user-specified granularity or DAY by default.

Step I.4. *Consider each binary fact type with an object type OT1 (a simple entity type, a temporal simple entity type, a complex entity type, or a temporal complex entity type).*

(a). If the OT1 type is associated with some structured type OT1_t, then the role of OT1 in the binary fact type is mapped to an attribute of the structured type OT*_t associated with the OT* type playing the other role in the binary fact type. The data type of the attribute is a collection type based on the structured type OT1_t if the role of OT1 is on the many side. Otherwise, the data type of the attribute is the structured type OT1_t. If the fact type is temporal, then the attribute is temporal.

(b). If the OT1 type is not associated with any structured type OT1_t, consider whether or not OT1 is a complex entity type or a temporal complex entity type whose sub schema has not yet been transformed. If yes, create a structured type OT1_t corresponding to the OT1 type and go back to the step (I.4.a). Otherwise, consider if OT1 is a temporal simple entity type. If yes, create a temporal structured type OT1_t corresponding to the OT1 type with the LIFE SPAN requirement and attributes mapped from the unique identifier of the OT1 type and go back to the step (I.4.a). Otherwise, the OT1 type is a non-temporal simple entity type. Check if the simple entity type OT1 is involved in at least one other many-to-one binary fact type in the main schema where the role of the OT1 type is on the many side. If yes, create a structured type OT1_t corresponding to the OT1 type. Map the unique identifier of OT1 to attributes of the structured type OT1_t and go back to the step (I.4.a). If not involved, the role of the simple entity type OT1 is mapped to an attribute of the structured type OT*_t. If the role of OT1 is on the many side, then the data type of the attribute is a collection type based on the data type DT of the unique identifier of OT1. Otherwise, the data type of the attribute is DT. If the fact type is temporal, then the attribute is temporal.

Step I.5. *Consider methods specified with the object type OT*.*

These methods are mapped to member functions/procedures of the structured type OT*_t. They are then coded in some procedural language allowed by the underlying DBMS.

9.2.2.2 The Transformation of a Main Schema

Once all sub schemas are successfully transformed, the transformation of a main schema is processed through steps II.1..II.5 as follows.

Step II.1. *Consider each reference type between a label type LT and a simple entity type ET. This reference type is used to uniquely identify entity instances of the ET simple entity type by instances of LT.*

If the entity type ET has already been associated with a structured type ET_t, go to the step (II.2). If the entity type ET is temporal, create a structured type ET_t corresponding to ET with the LIFE SPAN requirement and attributes mapped from the unique identifier LT. Otherwise, consider if the entity type ET is involved in at least one other many-to-one binary fact type in the main schema where the role of the entity type ET is on the many side, create a structured type ET_t corresponding to ET with attributes mapped from the unique identifier LT.

Step II.2. *Consider each binary fact type between two object types OT1 and OT2 where the object type (OT1 or OT2) can be a simple entity type, a complex entity type, a category type, a temporal simple entity type, a temporal complex entity type, or a temporal category type.*

(a). If both OT1 and OT2 are associated with two structured types OT1_t and OT2_t respectively, then map the role of the object type OT1 to an attribute of the structured type OT2_t and the role of the object type OT2 to an attribute of the structured type OT1_t. If the binary fact type is temporal, then these attributes are temporal. The data type of the attribute of the structured type OT2_t is a collection type of a reference type of the structured type OT1_t if the role of the object type OT1 is on the many side of the binary fact type. Otherwise, the data type is a reference type of the structured type OT1_t. Similarly for the data type of the attribute of the structured type OT1_t.

(b). If one of these two object types OT1 and OT2 is associated with a structured type, then let that object type be OT1 associated with the structured type OT1_t and the other

be OT2. That is, OT2 is a non-temporal simple entity type not involved in any many-to-one binary fact type in the main schema where the role of OT2 is on the many side. The role of OT2 is mapped to an attribute of the structured type OT1_t corresponding to the unique identifier of the entity type OT2. If the binary fact type is temporal, the attribute is temporal. The data type of the attribute is the data type specified for the unique identifier of the entity type OT2.

(c). If none of these two object types is associated with a structured type (i.e. these two object types OT1 and OT2 are non-temporal simple entity types), create a structured type ST corresponding to the binary fact type. If the binary fact type is temporal, the structured type ST is created with the LIFE SPAN requirement. Map the role of the object type OT1 to an attribute of the structured type ST corresponding to the unique identifier of the object type OT1 and the role of the object type OT2 to an attribute of the structured type ST corresponding to the unique identifier of the object type OT2. The data types of these attributes are the data types specified for their unique identifiers.

Step II.3. *Consider each nested fact type called a compound object type COT stemming from an n-ary fact type between n object types OTi for i in 1..n, n>=2.*

Create a structured type COT_t corresponding to the compound object type COT. If the object type COT is temporal, then the structured type COT_t is created with the LIFE SPAN requirement.

Consider each role of an object type OTi in the nested fact type. If the object type OTi is associated with a structured type OTi_t, map the role of OTi to an attribute of the structured type COT_t. The data type of the attribute is a reference type of the structured type OTi_t. Otherwise, i.e. OTi is a non-temporal simple entity type, map the role of OTi to an attribute of the structured type COT_t. The data type of the attribute is the data type specified for the unique identifier of the non-temporal simple entity type OTi.

Step II.4. *Consider each n-ary fact type named NFT between n object types OTi for i in 1..n, n>2 where the object type OTi can be a simple entity type, a complex entity type, a category type, a temporal simple entity type, a temporal complex entity type, or a temporal category type.*

Create a structured type NFT_t corresponding to the n-ary fact type NFT. If the fact type is temporal, the structured type NFT_t is created with the LIFE SPAN requirement.

Consider each role of an object type OTi in the n-ary fact type. If the object type OTi is associated with a structured type OTi_t, then map the role of OTi to an attribute of the structured type NFT_t. The data type of the attribute is a reference type of the structured type OTi_t. Otherwise, i.e. OTi is a non-temporal simple entity type, map the role of OTi to an attribute of the structured type NFT_t. The data type of the attribute is the data type specified for the unique identifier of the non-temporal simple entity type OTi.

Step II.5. *Consider each subtype/super type hierarchy between a super type OT and m sub types OTi for i in 1..m.*

At this step, all object types OT and OTi for i in 1..m have already been associated with structured types OT_t and OTi_t, respectively. Thus, the definition of each structured type OTi_t is expanded with the UNDER OT_t subtype clause for inheritance.

9.2.2.3 The Transformation of Constraints

Along with the transformation of the main and sub schemas, the mandatory and uniqueness constraints have been resolved at the implementation level while inherent constraints are checked during the parsing of the input TOONIAM conceptual schema. This subsection will take care of implicit constraints and explicit constraints that are declared with the specification in chapter 7. Nevertheless, it is bewared that not all of them can be automatically transformed. Other integrity constraints that cannot be automatically transformed must be tackled by application programs.

(a) Implicit Constraints

We introduce an automatic transformation of containment constraints and homogeneity constraints specified in chapter 7. The other implicit constraints (domain, occurrence frequency, exclusion, equality, subset, subtype exclusion, and subtype exhaustion constraints) have not yet been supported. Once implemented, those constraints are specified by using CHECK constraint declarations and/or triggers at the database level or by application programs at the application level.

(a.1) Containment constraints as conceptually shown in Figure 7.4

- The temporal object type O1 plays a role in a temporal unary fact type.

This case is considered in the sub schema that is used to detail the inner structure of the temporal object type O1. According to step I.3 in subsection 9.2.2.1, this role will

be mapped to a temporal attribute role_col of the structured type O1_t associated with the temporal object type O1. Let O1_tab be a temporal table defined on the O1_t structured type. Let role_col be the temporal column of the O1_tab table corresponding to the role_col attribute of the O1_t structured type. A containment constraint on the temporal unary fact type along with the temporal object type O1 is logically transformed as follows. If the returned value invalidRows is a positive number (>0), the constraint does not hold. Otherwise, the constraint is satisfied.

```
SELECT COUNT(*) INTO invalidRows
FROM O1_tab o
WHERE tDuring (VALIDTIME(o.role_col), LIFESPAN(o)) = 'TRUE';
```

- The temporal object type O1 plays a role in a temporal binary relationship type where the object type O2 plays the other role.

Similar to the previous case, let O1_t be the temporal structured type associated with O1. Let O1_tab be a temporal table defined on the O1_t structured type to keep instances of the O1_t structured type as tuples/rows in O1_tab. Let O2_col be the column of O1_tab corresponding to the O2_col attribute of the O1_t structured type. This O2_col column stems from the role of the O2 object type in the temporal binary relationship type. Therefore, the O2_col column is temporal. A containment constraint on the temporal binary relationship type along with the temporal object type O1 is logically transformed as follows. If the returned value invalidRows is a positive number (>0), the constraint does not hold. Otherwise, the constraint is satisfied.

```
SELECT COUNT(*) INTO invalidRows
FROM O1_tab o
WHERE tDuring (VALIDTIME(o.O2_col), LIFESPAN(o)) = 'TRUE';
```

- The temporal object type O1 plays a role in a temporal n-ary fact type for n > 2.

Let O1_t be the temporal structured type associated with O1. Let O1_tab be a temporal table defined on the O1_t structured type. Let NFT be the temporal n-ary fact type. Let NFT_t be the temporal structured type associated with the NFT fact type. Let NFT_tab be a temporal table defined on the NFT_t structured type. Let O1_col be a column in the temporal table NFT_tab corresponding to the O1_col attribute of the NFT_t structured type. This O1_col column stems from the role of the O1 object type in the temporal n-ary fact type NFT. Therefore, O1_col is a column each value of which is a reference to an object of the O1_t structured type, i.e. a tuple/row in the O1_tab table. A

containment constraint on the temporal n-ary fact type along with this temporal object type O1 is logically checked with the following SQL statement. If invalidRows is a positive number (>0), the constraint does not hold. Otherwise, the constraint is satisfied.

```
SELECT COUNT(*) INTO invalidRows
FROM NFT_tab n, O1_tab o
WHERE n.O1_col = o.ROWID AND tDuring (LIFESPAN(n), LIFESPAN(o)) = 'TRUE';
```

(a.2) Homogeneity constraints as conceptually shown in Figure 7.5

Let the object type O1 be the common object type of the two temporal relationship types that are being considered.

- If O1 is associated with a structured type O1_t, then the roles of O2 and O3 will be mapped to temporal attributes O2_col and O3_col, respectively, of the O1_t structured type. Let O1_tab be a temporal table defined on the structured type O1_t. The condition for a homogeneity constraint on the roles of the object type O1 is logically transformed into a following SQL statement. If invalidRows is a positive number (>0), the constraint does not hold. Otherwise, the constraint is satisfied.

```
SELECT COUNT(*) INTO invalidRows
FROM O1_tab o
WHERE tEqual (VALIDTIME(o.O2_col), VALIDTIME(o.O3_col)) = 'TRUE';
```

- If O1 is not associated with any structured type, then the role of O1 is mapped to a temporal attribute O1_col2 of a structured type O2_t associated with the relationship type or with the other object type O2, and also mapped to a temporal attribute O1_col3 of a structured type O3_t associated with the relationship type or with the other object type O3. Let O2_tab and O3_tab be temporal tables defined on the structured types O2_t and O3_t, respectively. The condition for a homogeneity constraint on the roles of the object type O1 is logically transformed into a following SQL statement. If invalidRows is a positive number (>0), the constraint does not hold. Otherwise, the constraint is satisfied.

```
SELECT COUNT(*) INTO invalidRows
FROM O2_tab o2, O3_tab o3
WHERE VALUE(o2.O1_col2) = VALUE(o3.O1_col3)
AND tEqual(VALIDTIME(o2.O1_col2), VALIDTIME(o3.O1_col3)) = 'TRUE';
```

(b) Explicit Constraints

Explicit constraints defined at the conceptual level using our explicit constraint specification are explicitly transformed into temporal SELECT statements. The WHERE clauses of these temporal SELECT statements are formulated using the syntax in

chapter 8. These SELECT statements can be stored in some system table so that at each commit time of any modification operation, these statements will be executed to select invalid rows in the database. If invalid rows exist, that modification operation will not be committed. That is, that modification operation is rejected. The formulation of a temporal SELECT statement corresponding to an explicit constraint is given below.

```
SELECT COUNT(*)
FROM <constrained_table_list>
WHERE NOT (<condition>);
```

In this template, <constrained_table_list> is a list of tables defined on the structured types obtained from the conceptual/logical transformation procedure. These tables are involved in the explicit constraint currently considered. <condition> is a corresponding condition of the explicit constraint where temporal operators G, F, X, P, and U are replaced by ALWAYS, ANYTIME, NEXT, PREV, and UNTIL, respectively. Also, Allen's temporal operators 'before', 'equal', 'meets', 'overlaps', 'during', 'starts', and 'finishes' are substituted by tbefore, tequal, tmeets, toverlaps, tduring, tstarts, and tfinishes.

9.2.3 The Temporal Object Relational Meta Database for 3D Objects

In order to obtain a temporal object relational meta database schema for temporal 3D objects, the conceptual/logical transformation procedure is applied to a TOONIAM conceptual meta schema for the representation of non-derived/derived non-temporal/temporal metadata about temporal 3D objects and their scenes. The temporal object relational meta database schema is composed of a collection of non-temporal/temporal structured types and constraint specifications. With the classification of metadata about temporal 3D objects in chapter 5 into non-explicit and explicit metadata, a temporal object relational meta database schema for temporal 3D objects consists of two parts: the first one fixed for content-based explicit metadata from any scene graph of an input 3D graphical data source and the second one changeable for non-explicit metadata up to further requirements of each individual application domain.

The fixed part of a temporal structured type associated with 3D scenes includes the LIFESPAN specification and the attribute that stems from the reference type for uniquely identifying each 3D scene in the input 3D graphical data source. The fixed part of each temporal structured type associated with 3D objects of the same type consists of the LIFESPAN specification, the attribute which stems from the reference type for uniquely

identifying each 3D object in a scene, the attribute which stems from the binary fact type for positions of each 3D object in a scene, and the attribute which stems from the binary fact type for the link between the metadata part and the graphical data part of a 3D object. This part appears in every temporal structured type definition associated with every 3D object type in a scene. So, these types and the tables defined on them only differ from each other in the changeable part for non-explicit metadata.

Besides, as 3D objects of the same type in a scene are represented by a temporal complex entity type, a temporal structured type is generated for this temporal complex entity type and a temporal typed table is defined on that temporal structured type as a meta table about those 3D objects. All non-explicit/explicit metadata about those 3D objects is gathered in one single meta table. So, all non-temporal/temporal non-explicit/explicit metadata about each temporal 3D object in a scene is kept in a single tuple/row of its corresponding meta table which is a temporal typed table. It is also made for an entire scene of the input 3D graphical data source and for categories of 3D objects of the same type. As a result, a temporal object relational meta database for temporal 3D objects will contain at least one meta table about 3D scenes, one meta table about 3D objects of each object type in a scene, and one meta table about 3D patterns (categories) of 3D objects of each object type in a scene.

For the case study, a temporal object relational meta database schema is reached for the period of time from 01/01/1965 to the infinity in Figure 9.4 where attributes stemming from relationship types representing non-derived metadata are shown in italics. Non-derived/derived metadata is then populated into meta tables that are typed tables of the temporal object relational meta database defined in Figure 9.5. By means of the temporal object relational SQL language proposed in chapter 8, the WITH LIFE SPAN clause is used to declare the life span requirement of a structured type and the VALIDTIME (granularity) clause to specify a temporal attribute of a structured type.

```
CREATE TYPE Region1Scene_t WITH LIFE SPAN(
        has_SceneID                 VARCHAR(50)
        ,is_created_by_Owner        VARCHAR(50));
CREATE TYPE Area_t WITH LIFE SPAN(
        has_ObjectID                VARCHAR(50)
        ,has_3DObject               REF(Transform) VALIDTIME(DAY)
        ,has_BoundingBox            MDSYS.SDO_GEOMETRY VALIDTIME(DAY)
        ,has_AreaCode               NUMBER
        ,belongs_to_AreaOwner       VARCHAR(50) VALIDTIME(DAY)
        ,MEMBER FUNCTION getLength RETURN NUMBER);
CREATE TYPE Building_t WITH LIFE SPAN(
```

```
        has_ObjectID                VARCHAR(50)
        ,has_3DObject               REF(Transform) VALIDTIME(DAY)
        ,has_BoundingBox            MDSYS.SDO_GEOMETRY VALIDTIME(DAY)
        ,has_BuildingName           VARCHAR(50)
        ,belongs_to_BuildingOwner   VARCHAR(50) VALIDTIME(DAY));
CREATE TYPE Road_t WITH LIFE SPAN(
        has_ObjectID                VARCHAR(50)
        ,has_3DObject               REF(Transform) VALIDTIME(DAY)
        ,has_BoundingBox            MDSYS.SDO_GEOMETRY VALIDTIME(DAY)
        ,has_RoadName               VARCHAR(50));
CREATE TYPE AreaModel_t(
        has_PatternID               REF(T3DModel_t)
        ,has_LandSort               VARCHAR(50) VALIDTIME(DAY));
CREATE TYPE BuildingModel_t(
        has_PatternID               REF(T3DModel_t)
        ,is_for_UsePurpose          VARCHAR(50) VALIDTIME(DAY));
CREATE TYPE RoadModel_t(
        has_PatternID               REF(T3DModel_t)
        ,is_for_VehicleKind         VARCHAR(50));
ALTER TYPE Region1Scene_t
ADD ATTRIBUTE contains_Area REF(Area_t) MULTISET VALIDTIME(DAY);
ALTER TYPE Area_t
ADD ATTRIBUTE belongs_to_Region1Scene REF(Region1Scene_t) VALIDTIME(DAY);
ALTER TYPE Region1Scene_t
ADD ATTRIBUTE contains_Road REF(Road_t) MULTISET VALIDTIME(DAY);
ATLER TYPE Road_t
ADD ATTRIBUTE belongs_to_Region1Scene REF(Region1Scene_t) VALIDTIME(DAY);
ALTER TYPE Region1Scene_t
ADD ATTRIBUTE contains_Building REF(Building_t) MULTISET VALIDTIME(DAY);
ALTER TYPE Building_t
ADD ATTRIBUTE belongs_to_Region1Scene REF(Region1Scene_t) VALIDTIME(DAY);
ALTER TYPE Area_t
ADD ATTRIBUTE is_categorized_by_AreaModel REF(AreaModel_t) VALIDTIME(DAY);
ALTER TYPE AreaModel_t
ADD ATTRIBUTE categorizes_Area REF(Area_t) MULTISET VALIDTIME(DAY);
ALTER TYPE Building_t
ADD ATTRIBUTE is_categorized_by_BuildingModel REF(BuildingModel_t) VALIDTIME(DAY);
ALTER TYPE BuildingModel_t
ADD ATTRIBUTE categorizes_Building REF(Building_t) MULTISET VALIDTIME(DAY);
ALTER TYPE Road_t
ADD ATTRIBUTE is_categorized_by_RoadModel REF(RoadModel_t) VALIDTIME(DAY);
ALTER TYPE RoadModel_t
ADD ATTRIBUTE categorizes_Road REF(Road_t) MULTISET VALIDTIME(DAY);
```

Figure 9.4 The Temporal Object Relational Meta Database Schema of the Case Study from 01/01/1965 to the Infinity

```
CREATE TABLE Region1Scene OF Region1Scene_t;
CREATE TABLE Area OF Area_t;
CREATE TABLE Building OF Building_t;
CREATE TABLE Road OF Road_t;
CREATE TABLE AreaModel OF AreaModel_t;
CREATE TABLE BuildingModel OF BuildingModel_t;
CREATE TABLE RoadModel OF RoadModel_t;
```

Figure 9.5 The Definitions of the Meta Tables in the Temporal Object Relational Meta Database of the Case Study

For explicit constraints, we show below the transformation of explicit constraints specified for the case study in subsection 7.2.3.2. However, the current implementation of this dissertation work has not yet been carried out with these constraints.

C13 – Each building must always be constructed totally on one area.

```
SELECT COUNT(*)
FROM Building i3, Area i2, Area i22
WHERE NOT (i2.REFC<>i22.REFC
        AND ALWAYS{on3D(i3.has_BoundingBox,i2.has_BoundingBox)='TRUE'}
        AND ALWAYS{on3D(i3.has_BoundingBox, i22.has_BoundingBox)='FALSE'})
```

C14 – The height of each building must be always greater than 10 meters.

```
SELECT COUNT(*)
FROM Building i3
WHERE NOT (ALWAYS{height3D(i3.has_BoundingBox)>10})
```

C15 – The school is expected to be built in front of at least one building.

```
SELECT COUNT(*)
FROM Building i3, Building i32
WHERE NOT (i32.has_BuildingName = 'School'
        AND i3.REFC <> i32.REFC
        AND front3D(i32.has_BoundingBox, i3.has_BoundingBox)='TRUE')
```

C16 – The school must be some time 500 meters near some road.

```
SELECT COUNT(*)
FROM Building i3, Road i4
WHERE NOT (i3.has_BuildingName = 'School'
        AND ANYTIME{distance3D(i3.has_BoundingBox, i4.has_BoundingBox)<500})
```

C17 – There must be at least one building constructed before the school is built.

```
SELECT COUNT(*)
FROM Building i3, Building i32
WHERE NOT (i32.has_BuildingName = 'School'
      AND i3.REFC<>i32.REFC
      AND tBEFORE (vtStart(i3.LIFESPAN), vtStart(i32.LIFESPAN)) = 'TRUE')
```

9.3 Summary

Regarding the proposed object relational database approach to dealing with the 3D graphical data part of 3D objects and scenes, some benefits are emphasized below.

- The reusability and sharability of existing 3D patterns are enabled by establishing a graphics library. The pre-processed graphics library helps the source of a 3D pattern to be examined once. Otherwise, the source of a 3D pattern might be loaded and processed many times whenever the pattern is involved in the pattern matching process during the data population.
- Due to the popularity and maturity of database technology, the querying over the 3D graphical data part of 3D objects and scenes is enabled in a standard way. Once queried, the 3D graphical data description of a part (i.e. component) or some 3D object, or of a 3D object, or of a few 3D objects (i.e. some part of a scene), or even of an entire scene can be obtained from the graphical database. It can be used for object/scene reconstruction at any detailed level from object to scene.

Regarding the proposed temporal object relational database approach to handling the metadata part of temporal 3D objects and scenes, some issues are discussed.

- It is claimed that the investigation of the proposed work into the temporal aspect of 3D objects and scenes is original in comparison with the related works. This support enriches the querying over 3D objects and their scenes with regard to valid time. Once 3D objects and scenes are time-varying, semantics/descriptive information about them, i.e. metadata, is also time-varying and might be collected and required along time. So, temporal data aggregation on such information is valuable for data analysis, decision support, and planning related to the 3D objects and scenes.
- An integration of non-temporal/temporal non-explicit/explicit metadata on temporal 3D objects and their scenes has been originally examined at both conceptual and logical data levels. Each individual semantic object type of 3D objects in a scene can be further defined for the representation of non-explicit metadata. Hence, non-explicit metadata can be conveniently gathered with explicit metadata that is directly extracted from an input 3D graphical data source. As compared to the related works, only this proposed work possesses this ability to produce the extensibility on metadata handling of temporal 3D objects and their scene towards particular application domains. This is because each particular application domain might include 3D objects with their various special non-presentable characteristics.

In summary, this chapter has specified a temporal object relational database for 3D objects in respect of valid time at both graphical data and metadata levels. The logical database schema is composed of an object relational graphical database schema and a temporal object relational meta database schema for the handling of the 3D graphical data and metadata parts of temporal 3D objects, respectively. The 3D graphical data part is shared by any application domain while the metadata part is specialized for each particular application domain. With this difference between these two parts, we deal with the 3D graphical data part at the logical level, i.e. at the implementation level, while the metadata part at the conceptual level, i.e. at the semantics level. Thus, the two corresponding schemas are also reached in a different manner. The resulting temporal object relational database schema for temporal 3D objects is now available for the temporal data population process which will be elaborated in the next chapter.

Chapter 10

The Proposed Temporal Data Population Process from a 3D Graphical Data Source with Valid Time

This chapter proposes a temporal data population process to populate graphical data and metadata about temporal 3D objects of a 3D graphical data source into an appropriate temporal object relational database. After the successful execution of this process, graphical data of these 3D objects is available in the graphical typed tables and metadata about these 3D objects is ready in the meta tables. The graphical typed tables and meta tables are defined in the previous chapter. They shape a temporal object relational database for temporal 3D objects handled in this proposed temporal object relational database system. As realized in chapter 9, only meta tables vary from application to application according to possible semantic object types present in each particular application domain.

10.1 The Preparation for the Proposed Temporal Data Population Process from a 3D Grpahical Data Source with Valid Time

Before going into detail over the data population processes of the graphical data and metadata parts of temporal 3D objects, the essential preparation for these two processes is taken into account as follows in order to prepare the graphics library and the input for the graphical data population process.

10.1.1 The Preparation Process

This preparation process will take into more consideration the format of an input 3D graphical data source. An overview of the preparation process is shown in Figure 10.1 where some specialized 3D object recognition process, the checking of the format of a 3D graphical data source, and the VRML converter are not detailed in this research.

The input of this preparation process is a 3D graphical scene contained in the input 3D graphical data source fed into the system with some period of time for validity.

If the scene is not formatted in the VRML language, a format conversion is needed by using some VRML converter. This is because the graphical data part handled in the

proposed system is supposed to be formatted in the VRML language. Therefore, a 3D scene graph in VRML is an expected output of this preparation process that will be next an input of the graphical data population process in subsection 10.2.

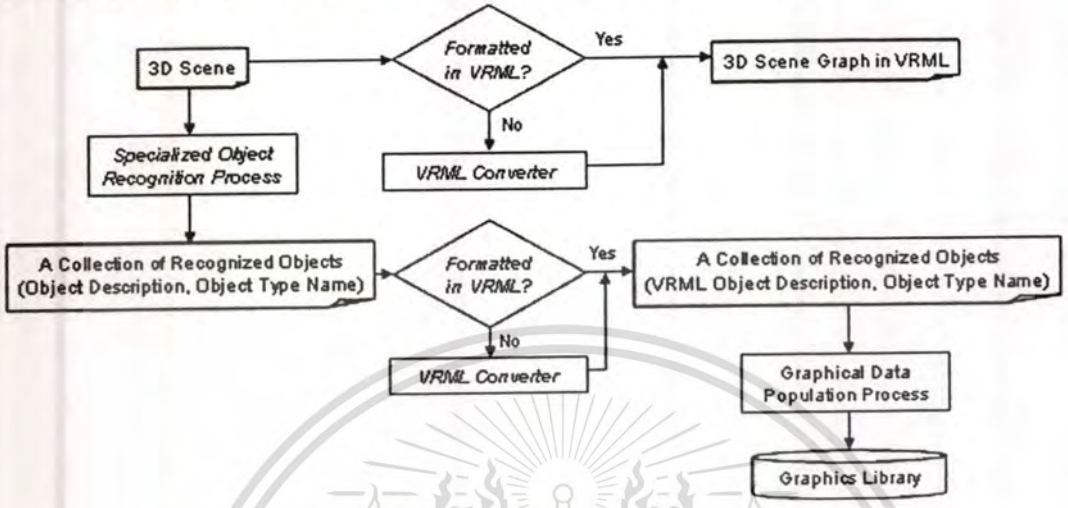


Figure 10.1 The Preparation Process

Moreover, some specialized object recognition process is invoked on the input 3D scene to recognize all meaningful objects included in the scene. This process is not included in this research. Nevertheless, a related topic can be found in [35] for cluttered scenes based on range images. After the successful execution of this process, a collection of recognized objects is obtained in such a way that each recognized object will be associated with its 3D graphical object description and a semantic object type name for the class of that object. If the 3D graphical object description is not formatted in the VRML language, some VRML converter is also employed to translate it into the one formatted in the VRML language. Thus, a final collection of recognized objects formatted in the VRML language is ready to be populated into the graphics library for the graphical data and metadata population processes.

Since those operations in the preparation process including a specialized 3D object recognition process, the checking of the format of a 3D graphical data source, and the format conversion process are excluded, this research is conducted with the two following assumptions that have been introduced in chapter 1.

Firstly, it is supposed that 3D objects and scenes are formatted in the VRML language regardless of how 3D objects and their scenes are created. This assumption does not narrow our work down because VRML is an open international standard for 3D

graphical data format and supported by many 3D authoring tools as their format output. Especially, most products related to VRML are free, open sourced such as VRML browsers. In case 3D objects are made in some special format different from VRML, a format converter to VRML is used. A VRML 3D object is visually distinguished from another one by its geometry and appearance such as color, texture, and so on.

Secondly, each 3D object is assumed to be created from some 3D pattern (template) in the library known to graphics designers. Patterns are also formatted in VRML. They can be obtained from specialized graphical object recognition and analysis methods as presented in Figure 10.1. Once a pattern is dragged from the library into the 3D scene that is being edited, a new 3D object will be created in the scene from that pattern and represented by a node of the Transform node type. The VRML description of the pattern will be wrapped into the children field of that node. The VRML field type of this children field is MFNode. In case the pattern is also a node of the Transform node type, the VRML description of the new 3D object is different from the one of the pattern in respect of transformation including translation, rotation, scale orientation, and scale. That is, the children field of the node describing the new 3D object and the one of the node describing the pattern are the same. Hence, only nodes of the Transform node type are checked to determine which part of an input scene graph describes some 3D object of interest. This point helps us reduce time and effort to conduct our temporal data population process. Regarding the relevance of this second assumption, it is thought that the assumption is acceptable because reuse is a current practice and a large number of VRML 3D models are available nowadays. However, a library of patterns is not always able to be prepared for the generating of 3D scenes. In such a context, 3D object creators and/or 3D scene creators, i.e. graphics designers, must know which objects their 3D scenes contain and which types these 3D objects are of. Using a bottom-up design methodology, individual 3D objects must be modeled first and then their 3D scenes are formed by laying out these 3D objects on purpose. So, graphics designers must be capable of supplying our system with so-called 3D patterns to establish a graphical library from those individual 3D objects. However, it is still arguable about the 3D scenes which are generated by some specialized 3D data modeling method with special equipments such as cameras and scanners instead of 3D

authoring tools and 3D programming toolkits. Although probably converted into VRML formatted files, 3D objects in these 3D scenes might be non-trivially recognized and distinguished from each other. Due to the key concentration of this research on temporal 3D object querying, we leave this issue as an open topic for the research arena of 3D object recognition over VRML 3D scenes.

10.1.2 The Handling of the Graphics Library

A so-called graphics library in the proposed system is a repository of 3D patterns used to determine which node of a Shape, Transform, Group, LOD, or Switch node type in a scene graph is used to describe some meaningful 3D object. If the node that is being checked is such a node, a semantic type of an extracted 3D object can be known from the information kept in the library associated with the 3D pattern of that 3D object.

As mentioned above, 3D patterns of the library are also stored in our Shape_tab, Transform_tab, Group_tab, Lod_tab, and Switch_tab graphical typed tables. This is because each 3D pattern can be a node of the Shape, Transform, Group, LOD, or Switch node type. All so-called VRML library files that contain those 3D patterns are parsed and processed only once. During the metadata population process, a VRML library file is not loaded and read for every check between the pattern contained in the library file and every node of the Shape, Transform, Group, LOD, or Switch node type in an input scene graph. As a result, we can avoid reading many times a VRML library file for a corresponding 3D pattern.

```
CREATE TYPE T3DModel_t(
        has_PatternURL VARCHAR(50)
        ,has_Gfx_Pattern node_t
        ,has_TypeName VARCHAR(50)
);
/
CREATE TABLE T3DModel OF T3DModel_t;
```

Figure 10.2 The T3DModel_t Structured Type and T3DModel Typed Table for the Handling of 3D Graphical Patterns in the Graphics Library

In order to link the information about each 3D pattern to its actual graphical description in a graphical typed table, we define the T3DModel_t structured type and the T3DModel typed table in Figure 10.2 for the handling of all 3D patterns in the library. Information about the VRML library file of each 3D pattern is kept in the has_PatternURL column. A reference to an actual 3D graphical description of a pattern in a graphical

typed table is stored in the has_Gfx_Pattern column. A value at this has_Gfx_Pattern column is an instance of the node_t data type according to the node type of the corresponding node that graphically represents the 3D pattern. A given semantic object type name associated with the pattern is a value at the has_TypeName column of the T3DModel typed table. Each instance of the T3DModel_t structured type, i.e. a tuple/row of the T3DModel table, corresponds to a 3D pattern that is in fact a category of a group of meaningful 3D objects of the same semantic object type. As shown in Figure 10.1, the graphical data population process in subsection 10.2 is also invoked for the populating of 3D patterns in VRML library files into the graphical typed tables and the T3DModel typed table of the graphics library.

Several 3D graphical patterns used in the case study are captured in the T3DModel typed table as shown in Table 10.1.

Table 10.1 The T3DModel Typed Table of the Case Study

REFC	has_PatternURL	has_Gfx_Pattern	has_TypeName
p1	pArea1_isb.wrl	node_t(0, SYS.ANYDATA.convertUROWID(shp1))	AREA
p2	pArea2_isb.wrl	node_t(0, SYS.ANYDATA.convertUROWID(shp2))	AREA
p3	pBuilding1_isb.wrl	node_t(0, SYS.ANYDATA.convertUROWID(shp3))	BUILDING
p4	pBuilding2_isb.wrl	node_t(1, SYS.ANYDATA.convertUROWID(tf3))	BUILDING
p5	pBuilding3_isb.wrl	node_t(1, SYS.ANYDATA.convertUROWID(tf5))	BUILDING
p6	pStreet1_isb.wrl	node_t(0, SYS.ANYDATA.convertUROWID(shp7))	ROAD
p7	pStreet2_isb.wrl	node_t(0, SYS.ANYDATA.convertUROWID(shp8))	ROAD
...	...	...	...

10.2 The Graphical Data Population Process

This graphical data population process is used to populate graphical data of a VRML input file into the graphical typed tables of the object relational graphical database defined in chapter 9. It is invoked on a VRML library file as well as on a VRML source file that contains a 3D graphical data source. The execution of this process is automatically accomplished regardless of how large the content of the VRML input file is. The overview on this graphical data population process is described in Figure 10.3.

In Figure 10.3, instances of the Shape, Transform, Group_t, Lod, and Switch structured types are generated from the definitions of nodes of the Shape, Transform, Group, LOD, and Switch node types in the input file, respectively. These instances are then inserted into the Shape_tab, Transform_tab, Group_tab, Lod_tab, and Switch_tab graphical typed tables with a check for no duplicate as noted earlier in subsection 9.1.3. The process is carried out within a bottom-up parsing process on the content of the

VRML input file so that non-decomposable Shape nodes can be reached prior to decomposable Transform, Group, LOD, and Switch nodes that might refer to other nodes via the fields of the MFNode field type. Such a chosen mechanism permits us to establish composition relationships among Shape, Transform, Group, LOD, and Switch nodes. After the graphical data population process, each node of the Shape, Transform, Group, LOD, and Switch VRML node types is stored in the corresponding Shape_tab, Transform_tab, Group_tab, LOD_tab, and Switch_tab graphical typed tables. In addition, each node of these VRML node types is associated with a REFC value of a tuple in a graphical typed table. Such a REFC value, called :nodeID, can be obtained via the RETURNING clause of an insertion. For no duplicate in the graphical typed tables, each insertion on any graphical typed table needs to check if an instance that is expected to be inserted exists.

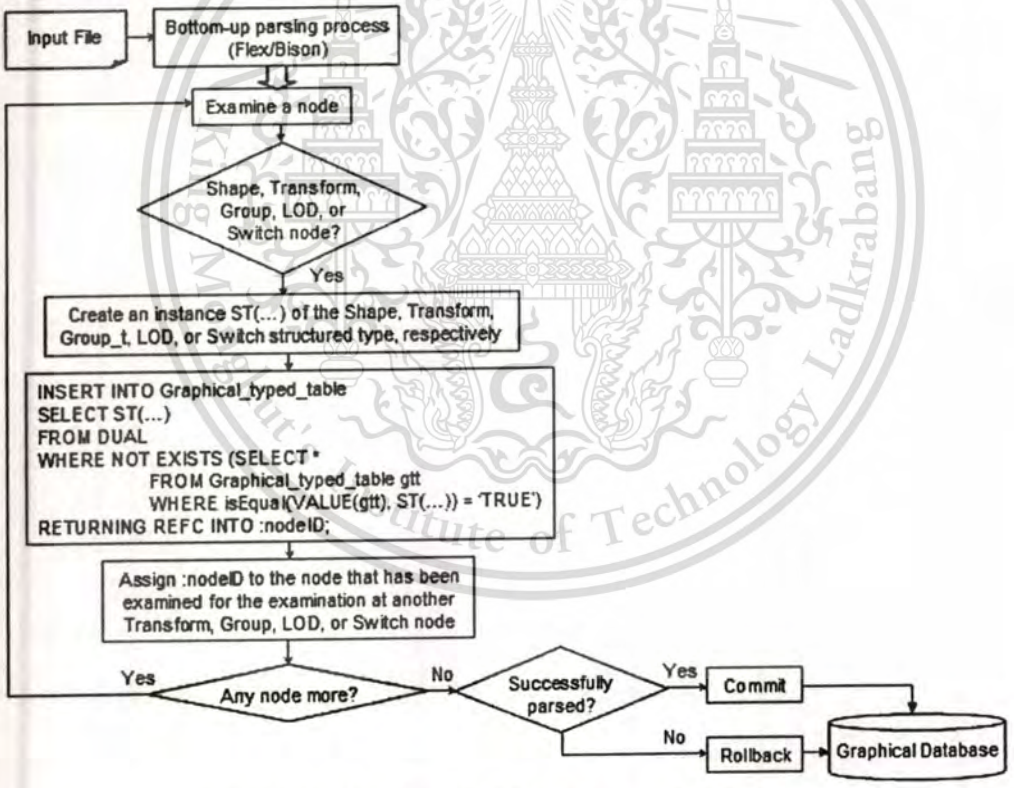


Figure 10.3 The Graphical Data Population Process

For example, we would like to insert into the Shape_tab table an instance Shape(...) of the Shape structured type for a node of the Shape VRML node type. An INSERT statement is issued below with our pre-defined isShapeEqual(shape1, shape2) function to check field-by-field the equality of two instances of the Shape structured type.

```
INSERT INTO Shape_tab
```

```
SELECT Shape(...)
FROM DUAL
WHERE NOT EXISTS (SELECT *
                  FROM Shape_tab t
                  WHERE isShapeEqual(VALUE(t), Shape(...)) = 'TRUE')
RETURNING REFC INTO :nodeID;
```

With the handling of 3D graphical data by means of OR database technology, the querying of 3D objects towards graphical perspectives is enabled with the use of the OR SQL language [17]. Moreover, the reconstructing of any VRML graphical node in the scene graph of a 3D graphical data source is trivially done from our graphical typed tables by using the toVRML() member function of a corresponding graphical structured type. This facility allows front-end applications to have access to some particular part of a 3D scene instead of the entire one by selecting several favorite 3D objects, extracting their VRML descriptions, and then doing post-processing in order to have a valid VRML output file interpretable to a VRML browser. It is also used to check the non-loss property of our graphical data population process as all 3D graphical data of VRML input files in respect of visualization can be totally restored from our graphical typed tables. For instance, we want to have the VRML description of a node of the Transform node type corresponding to the 'tf5' tuple in the Transform_tab graphical typed table. A SELECT statement is issued as follows to meet this requirement.

```
SELECT VALUE(t).toVRML() AS VRML_DESCRIPTION
FROM Transform_tab t
WHERE t.REFC = 'tf5';
```

After this stage, the content in the input file is re-formulated in terms of nodes of the Shape, Transform, Group, LOD, and Switch VRML node types. Each node is associated with a REFC value :nodeID of a corresponding tuple in a graphical typed table.

10.3 The Objectifying Process

As shown in Figure 5.1, the objectifying process will produce the input for our temporal metadata population process that is a scene graph described in terms of 3D objects instead of graphics elements. The main task of this process is to determine from which pattern a node of the Shape, Transform, Group, LOD, or Switch node type in the scene graph contained in the input source file is created. If such a pattern is found from the examination on the T3DModel table of the graphics library, the node is used to graphically represent a meaningful 3D object of a semantic type whose name is associated with the pattern. Furthermore, the objectifying process derives explicit

metadata about the object that has been found from the content of the 3D graphical data source. The content-based explicit metadata about each 3D object has been explained in subsection 5.3 and described in Figure 5.4. So, the output of this objectifying process is a scene graph composed of a collection of meaningful 3D objects. Each of such meaningful 3D objects, called GOi, is connected to the following explicit metadata.

- A pattern identifier which is a REFC value of a corresponding tuple/row in the T3DModel table: :patternID$_i$;
- An object identifier assigned to an associated graphical node of the Shape, Transform, Group, LOD, or Switch node type to uniquely identify GOi in the input scene: :objectID$_i$;
- An associated semantic object type name from the T3DModel table: :typename$_i$;
- A link to the graphical description of GOi in the graphical database: :gID$_i$;
- A minimum bounding box of the GOi object for positioning: :bbox$_i$;
- A set of components that form GOi in case the GOi object is a composite one, i.e. GOi is graphically represented by a node of the Transform, Group, LOD, or Switch node type: {:compID$_i$1, :compID$_i$2, ..., :compID$_i$k};
- The validity from the input given with the input source file that is a period of time [:IsStart, :IsEnd) where :IsEnd is not included into the validity.

Figure 10.4 summarizes the objectifying process as follows.

Firstly, the objectifying process runs a check on each node of the Shape, Transform, Group, LOD, or Switch node type in the scene graph that is now available in the graphical database. Each node of the Shape, Transform, Group, LOD, or Switch node type in the scene graph is associated with :nodeID$_i$ that is a REFC value of a corresponding tuple/row in a graphical typed table obtained from the graphical data population process in subsection 10.2. The check will determine the values of :patternID$_i$ and :typename$_i$ from the T3DModel typed table of the graphics library if the node that is being examined graphically represents some meaningful 3D object in the given scene. The following SELECT statement is used to assign a REFC value to :patternID$_i$ and a has_TypeName value to :typename from an appropriate tuple/row in the T3DModel typed table.

SELECT REFC, has_TypeName INTO :patternID$_i$, :typename$_i$
FROM T3DModel
WHERE isEqual(has_Gfx_Pattern, node_t(k, SYS.ANYDATA.convertUROWID(:nodeID$_i$))) = 'TRUE';

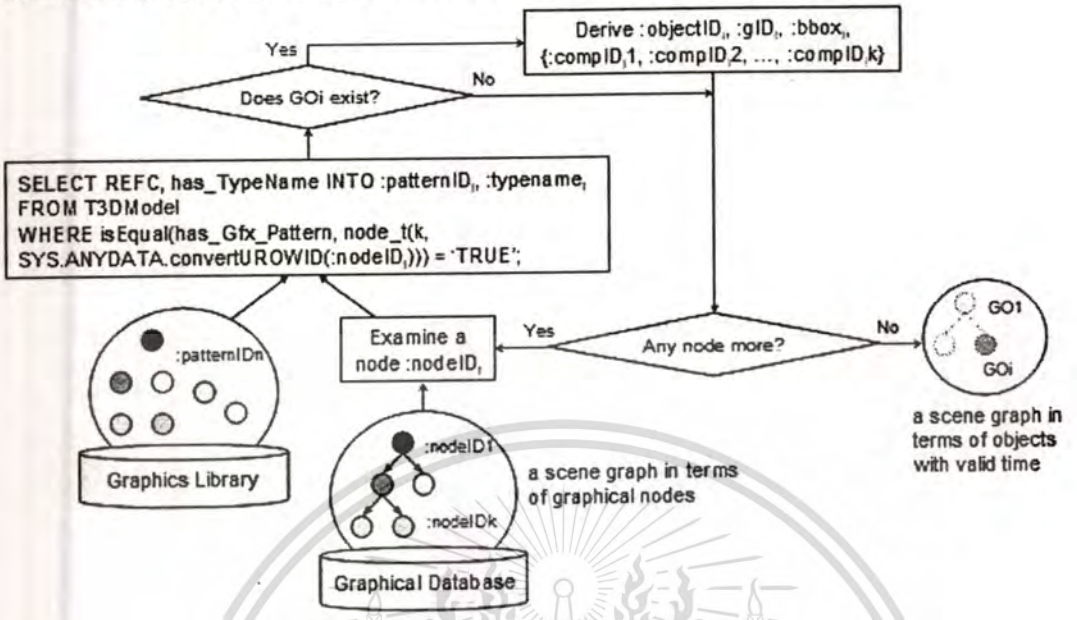


Figure 10.4 The Objectifying Process

The value has_Gfx_Pattern of a tuple/row in the T3DModel table is also an instance node_t(...) of the node_t type defined in subsection 9.1.1. The detail of our pre-defined isEqual() function is not shown here. If the REFC and has_TypeName values assigned to :patternID$_i$ and :typename$_i$ respectively exist, some 3D object GOi exists and is graphically represented by the graphical node that has a graphical description :nodeID$_i$ in the graphical database.

Secondly, once one 3D object GOi is determined, the values of :objectID$_i$, :gID$_i$, :bbox$_i$, and {:compID$_i$1, :compID$_i$2, ..., :compID$_i$k } are derived as follows whereas [:IsStart, :IsEnd) can be directly obtained from the input given along with the input source file.

- A value assigned to :objectID$_i$ is gained from the DEF declaration for the corresponding graphical node in the scene graph if specified; otherwise, the REFC value :nodeID$_i$ associated with GOi is used to form a value assigned to :objectID$_i$ as :objectID$_i$ = 'oid_'||:nodeID$_i$ where || is a string concatenation operator. It is realized that :objectID$_i$ is uniquely assigned to a GOi object. Also, :objectID$_i$ is time-invariant.
- A value assigned to :gID$_i$ is in fact :nodeID$_i$ to link to the graphical description of GOi in the graphical database. As determined in the graphical data population process, :nodeID$_i$ is the REFC value of a tuple/row of the graphical typed table defined on the

structured type corresponding to the graphical node type of the node that graphically represents GOi in the scene. Hence, :gID$_i$ is equal to :nodeID$_i$.

- A value assigned to :bbox$_i$ is computed from the VRML description of GOi by invoking the compBBox() member function of the structured type corresponding to the graphical VRML node type of the node that graphically represents GOi as shown in the following SELECT statement. The detail of this compBBox() member function will be elaborated in chapter 12.

```
SELECT VALUE(t).compBBox() INTO :bboxi
FROM Graphical_typed_table t
WHERE t.REFC = :nodeIDi;
```

- A {:compID$_i$1, :compID$_i$2, ..., :compID$_i$k} set of components in case the GOi object is composite is determined from the field of the MFNode field type of the Transform, Group, LOD, or Switch node graphically representing GOi. Each component of GOi is also a node of the Shape, Transform, Group, LOD, or Switch node type and has already been decided to graphically represent some 3D graphical object. Each :compID$_i$k value in the set is a REFC value of the tuple/row of the Shape_tab, Transform_tab, Group_tab, LOD_tab, or Switch_tab graphical typed table. Let :compIDs be {:compID$_i$1, :compID$_i$2, ..., :compID$_i$k}. A value assigned to :compIDs$_i$, i.e. {:compID$_i$1, :compID$_i$2, ..., :compID$_i$k}, is derived from either Transform_tab, Group_tab, LOD_tab, or Switch_tab graphical table by using one of the following SELECT statements up to the Transform, Group, LOD, or Switch node type of the node representing the GOi object, respectively. It is noted that there is a one-to-one correspondence between a :compID$_i$k value with some :objectID$_i$k value associated with some GOk object that is part of the GOi object. Therefore, each :compID$_i$k value in :compIDs$_i$ is replaced with a corresponding :objectIDk value and the composition relationships forming the GOi composite object have been established.

```
SELECT c.NodeID.AccessUROWID() BULK COLLECT INTO :compIDsi
FROM Transform_tab t, TABLE(t.children) c
WHERE t.REFC = :nodeIDi;

SELECT c.NodeID.AccessUROWID() BULK COLLECT INTO :compIDsi
FROM Group_tab t, TABLE(t.children) c
WHERE t.REFC = :nodeIDi;

SELECT c.NodeID.AccessUROWID() BULK COLLECT INTO :compIDsi
FROM LOD_tab t, TABLE(t.alevel) c
WHERE t.REFC = :nodeIDi;

SELECT c.NodeID.AccessUROWID() BULK COLLECT INTO :compIDsi
FROM Switch_tab t, TABLE(t.choice) c
WHERE t.REFC = :nodeIDi;
```

10.4 The Temporal Metadata Population Process

Different from the graphical data population process, the temporal metadata population process takes into consideration the temporal aspect of the input 3D graphical data source. This process takes place if and only if the graphical data population process is successfully finished with the input 3D graphical data source and the objectifying process is also accomplished to reformulate a scene graph in terms of 3D objects instead of graphics elements, i.e. graphical nodes. Thus, the objectifying process can be regarded as the preparation process for this process. This temporal metadata population process is sketched out in Figure 10.5 where two sub processes, namely the temporal metadata population sub process without change detection and the temporal metadata population sub process with change detection, are developed to perform the populating of non-temporal/temporal explicit metadata about temporal 3D objects into the temporal object relational meta database.

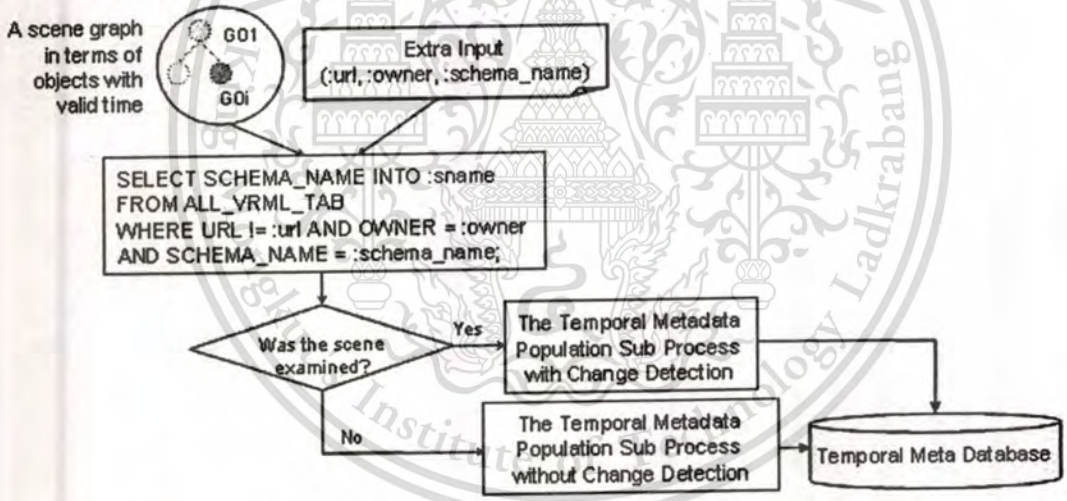


Figure 10.5 An Overview on the Temporal Metadata Population Process

Along with the input source file obtained after processed by the graphical data process and the objectifying process, this process requires extra input information including an owner name of the file, a database schema name for database connection, an optional scene name for the root node of the scene graph, a url (uniform resource locator) of the input source file, and some period of time for the validity of the content in the source file. In case no validity is specified, the period of time from NOW to the infinity is assumed. Metadata about 3D objects extracted from the input 3D graphical data

source is supposed to be true (valid) in the same periods of time. The input information related to one input source file is kept in our system table ALL_VRML_TAB below.

```
CREATE TABLE ALL_VRML_TAB (
        OWNER           VARCHAR2(30)    NOT NULL
        , SCHEMA_NAME   VARCHAR2(30)    NOT NULL
        , SCENE_NAME    VARCHAR2(30)
        , URL           VARCHAR2(500)   NOT NULL
        , VTSTART       DATE
        , VTEND         DATE
        , UNIQUE (OWNER, SCHEMA_NAME, URL));
```

The ALL_VRML_TAB table is also used to let us know whether there is a need of change detection of a scene in an F1.wrl input file for some period of time and itself in another F2.wrl input file for some other period of time that might overlap the period of time specified for the F1.wrl VRML input file. As shown in Figure 10.5, the checking of the need of change detection is done with the following SELECT statement. If a :sname result is NULL, metadata population is made with the temporal metadata population sub process without change detection for some periods of time in subsection 10.4.1. Otherwise, change detection along time is required and metadata population with respect to valid time is carried out with the temporal metadata population sub process with change detection in subsection 10.4.2.

```
SELECT SCHEMA_NAME INTO :sname
FROM ALL_VRML_TAB
WHERE OWNER = :owner AND SCHEMA_NAME = :schema_name AND URL != :url;
```

The output of this process is the non-temporal/temporal explicit metadata in the temporal meta database that is available for the querying of front-end applications. Other non-temporal/temporal non-explicit metadata is then populated according to each particular application domain by means of non-temporal/temporal INSERT and UPDATE statements.

Table 10.2 The ALL_VRML_TAB table for the case study

ROWID	OWNER	SCHEMA_NAME	SCENE_NAME	URL	VTSTART	VTEND
input1	TEST1	REGIONSCENE	Region1Scene	RegionView_isb_1955.wrl	01/01/1955	12/31/9999
input2	TEST1	REGIONSCENE	Region1Scene	RegionView_isb_1965.wrl	01/01/1965	12/31/9999
Input3	TEST1	REGIONSCENE	Region1Scene	RegionView_isb_1985.wrl	01/01/1985	12/31/9999

As for the case study, we keep the input for each process of a VRML input file in Table 10.2. At the first time, the input for the process on the RegionView_isb_1955.wrl VRML input file is given and stored in the *input1* row when none of the *input2* and *input3* rows exists. The metadata population from this VRML input file is done with the sub process without change detection in subsection 10.4.1. In contrast, the ones from the

two RegionView_isb_1965.wrl and RegionView_isb_1985.wrl VRML input files are done with the sub process with change detection in subsection 10.4.2.

10.4.1 The Temporal Metadata Population Sub Process without Change Detection

Without change detection, the main tasks of this sub process are listed below:

- to form a TOONIAM conceptual meta schema for temporal 3D objects and scene,
- to define non-temporal/temporal structured types and non-temporal/temporal typed tables as meta tables,
- to populate non-temporal/temporal explicit metadata into the meta tables by means of non-temporal/temporal INSERT statements. The explicit metadata about each 3D object stems from the extracted information (:patternID$_i$, :typename$_i$, :objectID$_i$, :gID$_i$, :bbox$_i$, {:compID$_i$1, :compID$_i$2, ..., :compID$_i$k}, and [:lsStart, :lsEnd)) associated with each 3D object determined in the previous objectifying process.

By following the description in subsection 7.2, the generation of a conceptual meta schema based on the proposed TOONIAM conceptual schema model in subsection 7.1 can be completed with a set of object types of 3D objects. We can obtain such a set from the :typename$_i$ value of each GOi object extracted in the previous objectifying process. Fact types, reference types, and operations (methods) for non-explicit metadata about these 3D objects and their categories can then be added into sub schemas of temporal complex entity types and of category types, respectively.

After the addition of the representation of non-temporal/temporal non-explicit metadata into the TOONIAM conceptual meta schema, the conceptual/logical transformation in subsection 9.2.2 is performed to define non-temporal/temporal structured types and non-temporal/temporal typed tables as meta tables.

Afterwards, only explicit metadata (:patternID$_i$, :typename$_i$, :objectID$_i$, :gID$_i$, :bbox$_i$, {:compID$_i$1, :compID$_i$2, ..., :compID$_i$k}, and [:lsStart, :lsEnd)) about each GOi object is populated into its corresponding meta table by means of non-temporal/temporal INSERT statements while non-explicit metadata can be updated later up to front-end application requirements. It is noted that one meta table is associated with one semantic object type and metadata about one 3D object is stored in one tuple of a meta table corresponding

to its semantic object type. Moreover, the name of the meta table is in fact the name of the semantic object type, which is the :typename$_i$ value extracted from the T3DModel typed table of the graphics library in the objectifying process.

For the case study, after the graphical data population process and the objectifying process, we achieve a few objects of the AREA and BUILDING semantic object types from the RegionView_isb_1955.wrl VRML input file for the period of time from 1955 to the infinity. The TOONIAM conceptual meta schema is established in Figures 7.7 (a), 7.8, 7.9 (a) and (b), 7.10 (a) and (b). The temporal meta database schema is achieved in Figure 9.4. Temporal metadata population is displayed in Tables 10.3, 10.4, and 10.5. As an illustration, the following INSERT statements are issued to populate metadata about one 3D object of the AREA semantic object type into the AREA meta table and its pattern into the AREAMODEL meta table where the values of the REFC columns are system-generated and non-explicit metadata is left as NULL.

```
INSERT INTO AREAMODEL (has_PatternID, has_LandSort)
SELECT :patternID_i, NULL
FROM DUAL
WHERE NOT EXISTS (SELECT *
                  FROM AREAMODEL
                  WHERE has_PatternID = :patternID_i)
RETURNING REFC INTO :areamodelID;

INSERT INTO AREA (has_ObjectID, has_3DObject, has_BoundingBox, has_AreaCode, belongs_to_AreaOwner, LIFESPAN,
belongs_to_Region1Scene, is_categorized_by_AreaModel)
VALUES (:objectID_i, :gID_i VALIDTIME PERIOD (:lsStart, :lsEnd), :bbox VALIDTIME PERIOD (:lsStart, :lsEnd), NULL,
NULL, PERIOD (:lsStart, :lsEnd), s1 VALIDTIME PERIOD (:lsStart, :lsEnd), :areamodelID VALIDTIME PERIOD (:lsStart,
:lsEnd))
RETURNING REFC INTO :areaID;
```

The supplementing of non-temporal/temporal non-explicit metadata can be carried out using non-temporal/temporal INSERT/UPDATE statements, respectively. For an illustration, non-temporal/temporal non-explicit metadata about areas can be supplied below from the description of the case study in chapter 6.

```
UPDATE AREA
SET has_AreaCode = 1,
        belongs_to_AreaOwner = 'Jeff' VALIDTIME PERIOD('01-01-1955', '12-31-9999')
WHERE REFC = 'a1';
```

Table 10.3 The REGION1SCENE temporal typed table as a meta table about regions after the process of the RegionView_isb_1955.wrl file for the period of valid time from 01/01/1955 to the infinity

REFC	has_SceneID	is_created_by_Owner	LIFESPAN	
			lsStart	lsEnd
s1	Region1	TEST1	01/01/1955	12/31/9999

Table 10.3 (cont.)

REFC	contains_Area			contains_Building		
	vtValue	ValidTime		vtValue	ValidTime	
		vtStart	vtEnd		vtStart	vtEnd
s1	{a1, a2, a3, a4}	01/01/1955	12/31/9999	{b1, b2}	01/01/1955	12/31/9999

Table 10.4 The AREA temporal typed table as a meta table about areas after the process of the RegionView_isb_1955.wrl file for the period of valid time from 01/01/1955 to the infinity

REFC	has_ ObjectID	has_3DObject			has_BoundingBox		
		vtValue	ValidTime		vtValue	ValidTime	
			vtStart	vtEnd		vtStart	vtEnd
a1	Parea1	tf6	01/01/1955	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.110025, 0, 31.620625, 42.942775, 0, 68.286375))	01/01/1955	12/31/9999
a2	Parea2	tf7	01/01/1955	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(42.44075, 0, 31.620625, 78.52325, 0, 68.286375))	01/01/1955	12/31/9999
a3	Parea3	tf8	01/01/1955	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(78.021075, 0, 31.620625, 103.604325, 0, 68.286375))	01/01/1955	12/31/9999
a4	Parea4	tf9	01/01/1955	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.10995, 0, 67.7843, 103.60445, 0, 92.7843))	01/01/1955	12/31/9999

Table 10.4 (cont.)

REFC	has_AreaCode	belongs_to_AreaOwner			LIFESPAN	
		vtValue	ValidTime		lsStart	lsEnd
			vtStart	vtEnd		
a1	1	Jeff	01/01/1955	12/31/9999	01/01/1955	12/31/9999
a2	2	Jane	01/01/1955	12/31/9999	01/01/1955	12/31/9999
a3	3	John	01/01/1955	12/31/9999	01/01/1955	12/31/9999
a4	4	Julia	01/01/1955	12/31/9999	01/01/1955	12/31/9999

Table 10.4 (cont.)

REFC	belongs_to_Region1Scene			is_categorized_by_AreaModel		
	vtValue	ValidTime		vtValue	ValidTime	
		vtStart	vtEnd		vtStart	vtEnd
a1	s1	01/01/1955	12/31/9999	am1	01/01/1955	12/31/9999
a2	s1	01/01/1955	12/31/9999	am1	01/01/1955	12/31/9999
a3	s1	01/01/1955	12/31/9999	am2	01/01/1955	12/31/9999
a4	s1	01/01/1955	12/31/9999	am1	01/01/1955	12/31/9999

Table 10.5 The BUILDING temporal typed table as a meta table about buildings after the process of the RegionView_isb_1955.wrl file for the period of valid time from 01/01/1955 to the infinity

REFC	has_ ObjectID	has_3DObject			has_BoundingBox		
		vtValue	ValidTime		vtValue	ValidTime	
			vtStart	vtEnd		vtStart	vtEnd
b1	Pbuilding1	tf10	01/01/1955	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(19.54722, 0, 38.2631, 39.5539774, 16.5003, 48.2631))	01/01/1955	12/31/9999
b2	Pbuilding2	tf11	01/01/1955	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2),	01/01/1955	12/31/9999

					SDO_ORDINATE_ARRAY(58.52348, 0, 37.9541, 71.18972, 13.00058, 56.1201))		

Table 10.5 (cont.)

REFC	has_BuildingName	belongs_to_BuildingOwner			LIFESPAN	
		vtValue	ValidTime		lsStart	lsEnd
			vtStart	vtEnd		
b1	Building1	Jeff	01/01/1955	12/31/9999	01/01/1955	12/31/9999
b2	Building2	Jane	01/01/1955	12/31/9999	01/01/1955	12/31/9999

Table 10.5 (cont.)

REFC	belongs_to_Region1Scene			is_categorized_by_BuildingModel		
	vtValue	ValidTime		vtValue	ValidTime	
		vtStart	vtEnd		vtStart	vtEnd
b1	s1	01/01/1955	12/31/9999	bm2	01/01/1955	12/31/9999
b2	s1	01/01/1955	12/31/9999	bm1	01/01/1955	12/31/9999

10.4.2 The Temporal Metadata Population Sub Process with Change Detection

Different from the previous sub process without change detection, this sub process takes place in the context of the existence of metadata about the same scene for some periods of valid time. In other words, the scene described by the input 3D graphical data source was examined. The main tasks of this sub process are listed as follows:

- to figure out the difference between the content in the F2.wrl input source file currently examined and the one in the F1.wrl input source file that is associated with some period of valid time overlapping the period of valid time given to the F2.wrl file;
- to update the TOONIAM conceptual meta schema and the temporal object relational meta database schema of the temporal object relational meta database;
- to populate new explicit metadata and changes of existing explicit metadata about 3D objects into the temporal object relational meta database.

As displayed in Figure 10.6, the input of the temporal metadata population sub process is the scene graph described in terms of objects with valid time that are produced by the previous objectifying process. Each object GOi is also associated with its extracted explicit metadata including :patternID$_i$, :typename$_i$, :objectID$_i$, :gID$_i$, :bbox$_i$, {:compID$_i$1, :compID$_i$2, ..., :compID$_i$k}, and [:lsStart, :lsEnd). Among these values, :objectID$_i$ is time-invariant in order to uniquely identify GOi of some semantic object type :typename$_i$ in the scene. :patternID$_i$ is used to refer to the 3D pattern associated with GOi during the time period [:lsStart, :lsEnd). :gID$_i$ is used to refer to the 3D graphical description (presentation) of GOi in the scene during the time period [:lsStart, :lsEnd).

:bbox$_i$ is used to refer to the position of GOi in the scene during the time period [:lsStart, :lsEnd). {:compID$_i$1, :compID$_i$2, ..., :compID$_i$k} is used to refer to the composition of GOi in the scene during the time period [:lsStart, :lsEnd).

Like the temporal metadata population sub process without change detection, this sub process populates temporal explicit metadata about temporal 3D objects by performing modifications on the existing temporal meta database for the time period [:lsStart, :lsEnd) as its output.

Based on what explicit metadata (:patternID$_i$, :typename$_i$, :objectID$_i$, :gID$_i$, :bbox$_i$, {:compID$_i$1, :compID$_i$2, ..., :compID$_i$k}, and [:lsStart, :lsEnd)) are extracted for each GOi object present in the scene of the input 3D graphical data source for the period of time from :lsStart to :lsEnd, change detections are determined for probable changes below.

(1) A check on :typename$_i$ to determine if the semantic object type of GOi is a new semantic object type for the time period [:lsStart, :lsEnd);

(2) A check on :objectID$_i$ to determine if GOi is a new object of the semantic object type whose name is :typename$_i$ for the time period [:lsStart, :lsEnd);

(3) A check on :patternID$_i$ to determine if GOi is associated with a different pattern for the time period [:lsStart, :lsEnd);

(4) A check on :gID$_i$ to determine if GOi has a different graphical description for the time period [:lsStart, :lsEnd);

(5) A check on :bbox$_i$ to determine if GOi changes its position for the time period [:lsStart, :lsEnd);

(6) A check on {:compID$_i$1, :compID$_i$2, ..., :compID$_i$k} to determine if GOi changes its composition for the time period [:lsStart, :lsEnd).

Beside the checks on extracted explicit metadata about 3D objects included in the scene for the time period [:lsStart, :lsEnd), the check (7) on existing explicit metadata about 3D objects handled in the temporal meta database is also run to determine which 3D objects are excluded from the scene the time period [:lsStart, :lsEnd).

(7) A check on :objectID$_i$ available in the temporal meta database to determine if a corresponding object GOj is absent from the scene for the time period [:lsStart, :lsEnd).

Using the database approach, these checks can be automatically accomplished by temporal SQL statements specified in chapter 8. It is noted that the check (1) is

performed at the type level while the others at the object (instance) level. Apart from these checks, a check on the life span of each GOi object will also be run so that the period of time from :lsStart to :lsEnd can be included into its life span.

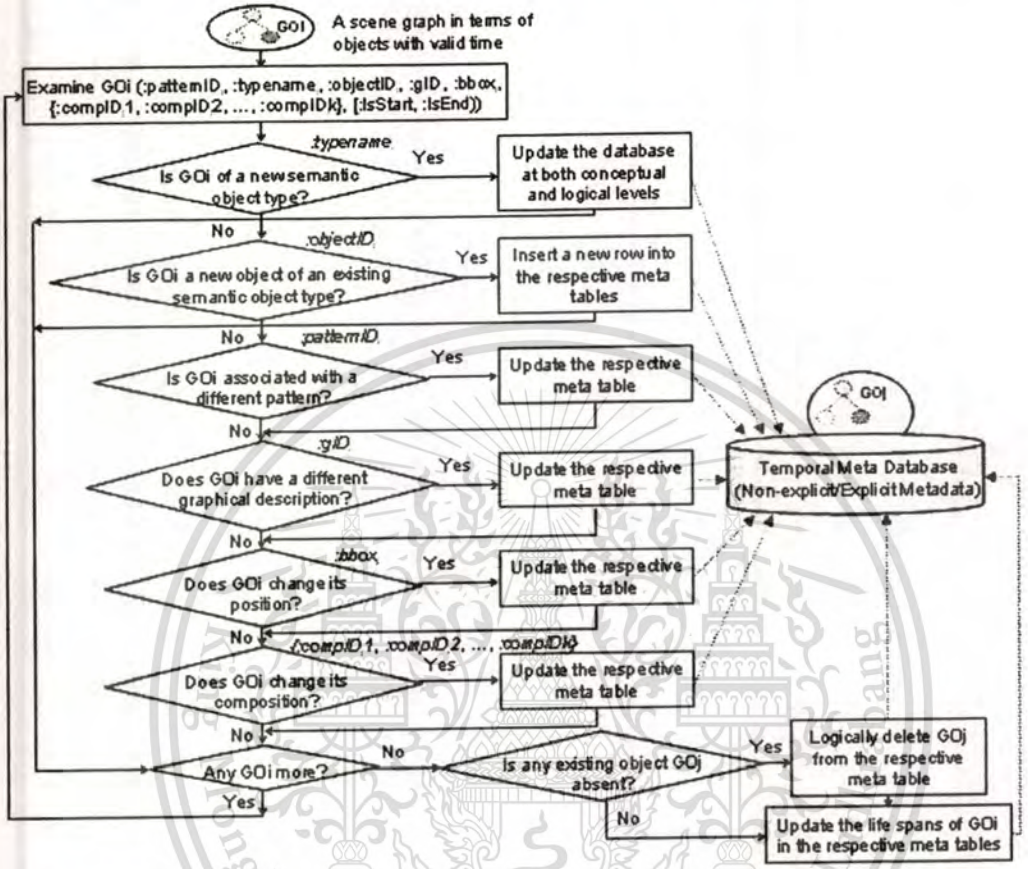


Figure 10.6 The Temporal Metadata Population Sub Process with Change Detection for the Period of Time [:lsStart, :lsEnd)

The following will show the details of this sub process for [:lsStart, :lsEnd).

(1) A check on :typename$_i$

As noted earlier, the name :typename$_i$ of the semantic object type of GOi is conventionally used as the name of the meta table where the metadata about GOi is gathered in a corresponding row/tuple. Therefore, running a check on :typename$_i$ to determine if the semantic object type of GOi is a new semantic object type in the scene for the period of time from :lsStart to :lsEnd is in fact running a check on whether any meta table whose name is :typename$_i$ exists in the temporal meta database. In order to check the existence of a table in general, an examination on an appropriate data dictionary view is taken. For example, with an implementation on Oracle 10g/11g, the

following SQL statement can be issued against the ALL_OBJECT_TABLES view with the value :owner provided as an extra input.

```
SELECT COUNT(*) INTO :isNew
FROM ALL_OBJECT_TABLES
WHERE OWNER = :owner AND TABLE_NAME = :typename;;
```

If the returned value :isNew is one (1), the table named :$typename_i$ exists. In other words, the corresponding semantic object type has existed in the scene for some period of time. So, proceed with the check (2) on :$objectID_i$.

If :isNew is zero (0), there is no table named :$typename_i$ in the temporal meta database. That is, the corresponding semantic object type has not yet existed in the scene for any period of time. Some updates are made as follows on the temporal meta database at both conceptual and logical levels and then move back to the check (1) for another object GOi.

- At the conceptual level, the main schema of the TOONIAM conceptual meta schema is updated with a new temporal complex entity type OT_i corresponding to the new semantic object type of GOi and a category type CT_i associated with this new temporal complex entity type. A one-to-many temporal binary fact type is established between the temporal complex entity type OT_0 corresponding to the semantic object type of the scene and the new temporal complex entity type OT_i for the fact "A scene contains one or many objects of OT_i". A many-to-one temporal binary fact type is also established between the new temporal complex entity type OT_i and its associated category type CT_i for the fact "One category of CT_i categorizes one or many objects of OT_i". If GOi is a composite object, i.e. {:$compID_i1$, :$compID_i2$, ..., $compID_ik$} is not an empty set, let :$typename_i1$, :$typename_i2$, ..., :$typename_ik$ be the names of the semantic object types of GO_i1, GO_i2, ..., GO_ik, associated with :$compID_i1$, :$compID_i2$, ..., :$compID_ik$, respectively, that form the GOi composite object. If any GO_i1, GO_i2, ..., or GO_ik has not yet been examined, an examination on each of them is taken to make sure that each semantic object type named :$typename_i1$, :$typename_i2$, ..., or :$typename_ik$ has already been associated with some temporal complex entity type OT_i1, OT_i2, ..., or OT_ik. A one-to-many temporal binary fact type is established between the new temporal complex entity type OT_i and each of the temporal complex entity types OT_i1, OT_i2, ..., OT_ik to represent the fact "One object of OT_i is composed of one or many objects of OT_i1, OT_i2,

..., or OT_ik", respectively. At the sub schema level, the sub schemas of the new temporal complex entity type OT_i and its associated category type CT_i are defined according to the specification in subsection 7.2.2. Additional fact types, reference types, and operations (methods) for non-temporal/temporal non-explicit metadata about 3D objects of this new semantic object type and their categories, i.e. instances of the category type CT_i, might be included in their corresponding sub schemas.

- At the logical level, a temporal structure type OT_i_t is defined corresponding to the new temporal complex entity type OT_i according to the I. steps in subsection 9.2.2.1. Also, a structured type CT_i_t is defined corresponding to the new category type CT_i following the I. steps in subsection 9.2.2.1. Relationships among the structured type OT_i_t and the one OT_0_t corresponding to the temporal complex entity type OT_0 for the entire scene, among the structured type OT_i_t and the one CT_i_t are defined by using the II.2.a step in subsection 9.2.2.2. Temporal typed tables are then defined on these structured types using the specification of the temporal object relational SQL language proposed in chapter 8. The temporal typed table that is a meta table about objects of the semantic object type named :$typename_i$ is also named :$typename_i$ and the temporal typed table that is a meta table about categories of the objects of the semantic object type named :$typename_i$ is named :$typename_i$||'Model' where '||' is the string concatenation operator.

- Afterwards, the explicit metadata population into the new meta table is carried out with the extracted metadata associated with the GOi object by using the non-temporal/temporal INSERT statements like those in the previous temporal metadata population sub process without change detection. Non-explicit metadata is left as NULL in those INSERT statements and can later be updated by proper non-temporal/temporal INSERT and/or UPDATE statements if required.

(2) A check on :$objectID_i$

For this check, the meta table about 3D objects of the semantic object type whose name is :$typename_i$ exists. Let meta_tab be that meta table and the name of meta_tab is conventionally :$typename_i$. An examination on meta_tab is taken to determine if GOi is a new object of the semantic object type whose name is :$typename_i$ for the period of time from :lsStart to :lsEnd with the following SELECT statement.

```
SELECT COUNT(*) INTO :isNew
FROM meta_tab
```

```
WHERE has_ObjectID = :objectID_i;
```

If the returned value :isNew is one (1), GOi exists in the scene for some period of time. So, the next check (3) is considered.

If the returned value :isNew is zero (0), GOi has not yet existed in the scene for any period of time. Only modifications at the instance (data) level are made on the respective meta tables and then move back to the check (1) for another object GOi. The modifications include a temporal insertion into meta_tab where non-explicit metadata is left as NULL, an insert into the meta_model_tab table associated with the CT_i category type, and an update on the meta table about the entire scene to express the fact that the scene contains the GOi object during the period of time from :lsStart to :lsEnd. These modification statements are similar to those in the temporal metadata population sub process without change detection.

(3) A check on :patternID$_i$

A check on :patternID$_i$ aims to determine if GOi is associated with a different pattern for the period of time from :lsStart to :lsEnd. In this check, GOi has already existed in the scene for some period of time that might overlap the period of time [:lsStart, :lsEnd).

At first, a check on whether the metadata about the pattern identified by :patternID$_i$ exists in its corresponding meta table meta_model_tab is performed. If not yet, an insertion on meta_model_tab is needed with NULL values for non-explicit metadata about the pattern.

```
INSERT INTO meta_model_tab (has_PatternID, ... )
SELECT :patternID_i, ...
FROM DUAL
WHERE NOT EXISTS (SELECT * FROM meta_model_tab WHERE has_PatternID = :patternIDi);
```

After that, this check is run with the following temporal SELECT statement using the ALWAYS temporal operator defined in chapter 8.

```
SELECT COUNT(*) INTO :isChanged
FROM meta_tab
WHERE has_ObjectID = :objectID_i
AND ALWAYS PERIOD (:lsStart, :lsEnd) {is_categorized_by_typenameModel = (SELECT REFC
                                                                          FROM meta_model_tab
                                                                          WHERE has_PatternID = :patternID_i)};
```

If the returned value :isChanged is one (1), GOi has already associated with the pattern identified by :patternID$_i$ for the period of time from :lsStart to :lsEnd. So, proceed with the next check (4) on :gID$_i$.

If the returned value :isChanged is zero (0), update the metadata about GOi with the pattern identified by :patternID$_i$ for the period of time from :lsStart to :lsEnd by means of the following temporal UPDATE statement. Then, move to the next check (4) on :gID$_i$.

```
UPDATE meta_tab
SET is_categorized_by_typenameModel = (SELECT REFC
                                       FROM meta_model_tab
                                       WHERE has_PatternID = :patternID_i) VALIDTIME PERIOD (:lsStart, :lsEnd)
WHERE has_ObjectID = :objectID_i;
```

(4) A check on :gID$_i$

A change on the pattern will lead to a change on the graphical description of GOi. In contrast, a change on the graphical description of GOi might not make any change on the pattern associated with GOi. Therefore, this check is run on the value :gIDi of GOi to determine if the graphical description of GOi is changed to the one identified by :gIDi for the period of time from :lsStart to :lsEnd. The following temporal SELECT statement is issued.

```
SELECT COUNT(*) INTO :isChanged
FROM meta_tab
WHERE has_ObjectID = :objectID_i
AND ALWAYS PERIOD (:lsStart, :lsEnd) {has_3DObject = :gID_i};
```

If the returned value :isChanged is one (1), GOi is graphically presented in the scene by the graphical description identified by :gID$_i$ for the period of time from :lsStart to :lsEnd. So, move to the next check (5) on :bbox$_i$.

If the returned value :isChanged is zero (0), no metadata about GOi in meta_tab exists to say that GOi is graphically presented in the scene by the graphical description identified by :gIDi for the period of time from :lsStart to :lsEnd. Thus, a temporal UPDATE statement is issued below to capture such information. After that, move to the check (5).

```
UPDATE meta_tab
SET has_3DObject = :gID_i VALIDTIME PERIOD (:lsStart, :lsEnd)
WHERE has_ObjectID = :objectID_i;
```

(5) A check on :bbox$_i$

A change on its position in the scene leads to a change on the minimum bounding box of GOi. Thus, a check on :bbox$_i$ of GOi with the following temporal SELECT statement is used to determine if GOi changes its position in the scene for the period of time from :lsStart to :lsEnd.

```
SELECT COUNT(*) INTO :isChanged
FROM meta_tab
WHERE has_ObjectID = :objectID_i
AND ALWAYS PERIOD (:lsStart, :lsEnd) {isEqual(has_BoundingBox, :bbox_i) = 'TRUE'};
```

If the returned value :isChanged is one (1), the minimum bounding box of GOi is :bbox$_i$ for the period of time from :lsStart to :lsEnd. So, proceed with the next check (6).

If the returned value :isChanged is zero (0), no information in meta_tab says that the minimum bounding box of GOi is :bbox$_i$ for the period of time [:lsStart, :lsEnd). So, update the minimum bounding box of GOi with :bbox$_i$ for the period of time [:lsStart, :lsEnd); i.e., the position of GOi in the scene for that period of time is identified by :bbox$_i$.

```
UPDATE meta_tab
SET has_BoundingBox = :bboxi VALIDTIME PERIOD (:lsStart, :lsEnd)
WHERE has_ObjectID = :objectIDi;
```

(6) A check on {:compID$_i$1, :compID$_i$2, ..., :compID$_i$k}

Different from the checks (2..5), this check involves not only the meta table meta_tab about 3D objects of the semantic object type whose name is :typename$_i$ but also the meta tables meta_tab$_i$1, meta_tab$_i$2, ..., and meta_tab$_i$k about 3D objects of the semantic object types whose names are :typename$_i$1, :typename$_i$2, ..., and :typename$_i$k that form composite objects of the semantic object type whose name is :typename$_i$. GOi is among such composite objects and composed of those objects identified by :compID$_i$1, :compID$_i$2, ..., and :compID$_i$k. It is also noted that these type names :typename$_i$1, :typename$_i$2, ..., and :typename$_i$k might be duplicate. Thus, let :typename$_i$k1, :typename$_i$k2, ..., :typename$_i$kk be among :typename$_i$1, :typename$_i$2, ..., and :typename$_i$k where :typename$_i$k1, :typename$_i$k2, ..., and :typename$_i$kk are different from each other. For the type name :typename$_i$k1, a group {:compID$_{i,k1}$1, compID$_{i,k1}$2, ..., } is created where compID$_{i,k1}$1, compID$_{i,k1}$2, ..., are among :compID$_i$1, :compID$_i$2, ..., :compID$_i$k and of the same semantic object type whose name is :typename$_i$k1. Similar consideration is made for :typename$_i$k2, ..., :typename$_i$kk.

Firstly, we examine meta_tab for the fact that "GOi is composed of compID$_{i,k1}$1, compID$_{i,k1}$2, ... that are of the same semantic object type whose name is :typename$_i$k1 for the period of time [:lsStart, :lsEnd)" using the following temporal SELECT statement.

```
SELECT COUNT(*) INTO :isChangedk1
FROM meta_tab
WHERE has_ObjectID = :objectIDi
AND ALWAYS PERIOD(:lsStart, :lsEnd) { is_composed_of_typenamei k1 = (:compIDi,k1 1, :compIDi,k1 2, ...) };
```

If the returned value :isChangedk1 is zero (0), no metadata about the fact that "GOi is composed of {:compID$_{i,k1}$1, :compID$_{i,k1}$2, ...} of the same semantic object type whose

name is :typename$_i$1 for the period of time from :lsStart to :lsEnd" is recorded. Therefore, a temporal UPDATE statement is issued against the meta table meta_tab as follows.

```
UPDATE meta_tab
SET is_composed_of_typename_i_k1 = (:compID_i,k1_1, :compID_i,k1_2, ...) VALIDTIME PERIOD (:lsStart, :lsEnd)
WHERE has_ObjectID = :objectID_i;
```

Similarly, a check is made on the columns is_composed_of_typename$_i$k2, ..., is_composed_of_typename$_i$kk of the meta table meta_tab for the other type names :typename$_i$k2, ..., :typename$_i$kk.

Secondly, we examine meta_tab$_i$k1, meta_tab$_i$k2, ..., and meta_tab$_i$kk for the facts that "{:compID$_{i,k1}$1, :compID$_{i,k1}$2, ...} of the same semantic object type whose name is :typename$_i$k1 form the GOi composite object for the period of time [:lsStart, :lsEnd)", "{:compID$_{i,k2}$1, :compID$_{i,k2}$2, ...} of the same semantic object type whose name is :typename$_i$k2 form the GOi composite object for the period of time [:lsStart, :lsEnd)", ..., and "{:compID$_{i,kk}$1, :compID$_{i,kk}$2, ...} of the same semantic object type whose name is :typename$_i$kk form the GOi composite object for the period of time [:lsStart, :lsEnd)".

Consider meta_tab$_i$k1, the following temporal SELECT statement is issued.

```
SELECT COUNT(*) INTO :isChanged_k1 1
FROM meta_tab_i k1
WHERE has_ObjectID = :compID_i,k1 1
AND ALWAYS PERIOD (:lsStart, :lsEnd) {makes_up_typename_i = :objectID_i};

SELECT COUNT(*) INTO :isChanged_k1 2
FROM meta_tab_i k1
WHERE has_ObjectID = :compID_i,k1 2
AND ALWAYS PERIOD (:lsStart, :lsEnd) {makes_up_typename_i = :objectID_i};
...
```

If any of the returned values :isChanged$_{k1}$1, :isChanged$_{k1}$2, ... is zero (0), there is no metadata about the fact that ":compID$_{i,k1}$1, or :compID$_{i,k1}$2, or ... of the same semantic object type whose name is :typename$_i$k1 form the GOi composite object for the period of time [:lsStart, :lsEnd)". So, a temporal UPDATE statement is used to update meta_tab$_i$k1.

```
UPDATE meta_tab_i k1
SET makes_up_typename_i = :objectID_i VALIDTIME PERIOD (:lsStart, :lsEnd)
WHERE has_ObjectID IN (:compID_i,k1 1, :compID_i,k1 2, ...);
```

Similarly, a check is made for each of the other meta tables meta_tab$_i$k2, ..., meta_tab$_i$kk corresponding to the semantic object types whose names are :typename$_i$k2, ..., :typename$_i$kk.

Thirdly, we check any object GOi more in the returned set of 3D objects extracted by the objectifying process. If any, proceed with check (1). If no, move to check (7).

(7) A check on :objectID$_j$, which is available in the temporal meta database

After the examination on every piece of extracted explicit metadata about each GOi object in the scene for the period of time from :lsStart to :lsEnd, a check on :objectID$_j$, which is available in the temporal meta database, is run to determine if any existing object of some existing semantic object type identified by :objectID$_j$ is absent from the scene being considered for the period of time from :lsStart to :lsEnd. For this change detection, let a set of objects extracted by the previous objectifying process be { ..., :objectID$_i$, ...}, let meta_tab* be any meta table about 3D objects of some semantic object type recorded in the temporal meta database. The following SELECT statement is issued on meta_tab* because we do not know about the semantic object type of objects that might be excluded from the scene for the period of time from :lsStart to :lsEnd.

```
SELECT REFC BULK COLLECT INTO :refc_s
FROM meta_tab*
WHERE has_ObjectID NOT IN (..., :objectID_i, ...);
```

The result of this statement is a set :refc_s of REFC values. Each REFC value in :refc_s is associated with some 3D object. The cardinality of the set can be from 0 to some positive integer number n. In turn, for each REFC value in :refc_s, we logically delete metadata about a corresponding 3D object from its corresponding meta table to say that the 3D object is not present in the scene for the period of time from :lsStart to :lsEnd. A temporal DELETE statement with the VALIDTIME clause is stated as follows:

```
DELETE FROM meta_tab*
VALIDTIME PERIOD (:lsStart, :lsEnd)
WHERE REFC IN :refc_s;
```

We remove the information about the fact that the scene contains the 3D object with a following ADELETE statement with the period of time from :lsStart to :lsEnd particularly for the 3D object. Let scene_meta_tab be the meta table about the entire scene and contains_typename be the temporal attribute stemming from the fact type "Scene contains objects of the semantic object type :typename for some periods of time".

```
ADELETE FROM scene_meta_tab
ATTRIBUTES (contains_typename VALIDTIME PERIOD (:lsStart, :lsEnd) WHERE contains_typename IN :refc_s)
WHERE is_created_by_Owner = :owner AND is_under_Schema = :schema_name;
```

Finally, we update the life spans of each GOi object in the meta table meta_tab and the life span of the corresponding entire scene in the meta table scene_meta_tab. Our pre-defined tUNION() function is used to calculate a union of two sets of disjoint periods of time. The output of this function is also a set of disjoint periods of time.

```
UPDATE meta_tab
SET LIFESPAN = tUNION(LIFESPAN, PERIOD(:lsStart, :lsEnd))
```

```
WHERE has_ObjectID = :objectID;;

UPDATE scene_meta_tab
SET LIFESPAN = tUNION(LIFESPAN, PERIOD(:lsStart, :lsEnd))
WHERE is_created_by_Owner = :owner AND is_under_Schema = :schema_name;
```

For a demonstration with the case study, we consider the process on the RegionView_isb_1965.wrl VRML input file for a given period of time from 01/01/1965 to the infinity that is invoked after the one on the RegionView_isb_1955.wrl VRML input file for a given period of time from 01/01/1955 to the infinity. At the conceptual level, the TOONIAM conceptual meta schema for the case study is affected and changed with a new main schema shown in Figure 7.7 (b) and a new sub schema defined for the new Road temporal complex entity type in Figure 7.9 (c). Corresponding to the changes on the conceptual meta schema, the meta database schema is changed with the new temporal structured types ROAD_t and ROADMODEL_t and an alteration on the REGION1SCENE_t temporal structured type with a new contains_Road temporal attribute. So, a new ROAD temporal typed table as a meta table about roads is created on the ROAD_T temporal structured type and a new contains_Road temporal column is added into the REGION1SCENE temporal typed table. Also, a new ROADMODEL temporal typed table as a meta table about road categories is defined on the ROADMODEL_t temporal structured type. Changes at the instance level are also made on the REGION1SCENE and AREA temporal typed tables. Metadata about roads and metadata changes on existing meta tables are shown in Tables 10.6, 10.7, and 10.8.

Different from the process on the RegionView_isb_1965.wrl VRML input file for a given period of time from 01/01/1965 to the infinity, the process on the RegionView_isb_1985.wrl VRML input file for a given period of time from 01/01/1985 to the infinity does not make any change on the conceptual meta schema; thus no change on the meta database schema. Only changes at the instance level take place. They include metadata changes on the meta tables about the entire scene, areas, and buildings as displayed in Tables 10.9..10.11.

Metadata about the categories of areas, buildings, and roads are given in the AreaModel, BuildingModel, and RoadModel temporal meta tables as shown in Tables 10.12, 10.13, and 10.14 for the period of valid time from 01/01/1985 to the infinity. In these tables, non-explicit metadata about the categories is not available in the temporal meta database; thus, can be supplemented later according to domain requirements.

Table 10.6 A part of the REGION1SCENE temporal typed table as a meta table about regions after the process of the RegionView_isb_1965.wrl file for the period of valid time from 01/01/1965 to the infinity

REFC	contains_Area			contains_Building			contains_Road		
	vtValue	ValidTime		vtValue	ValidTime		vtValue	ValidTime	
		vtStart	vtEnd		vtStart	vtEnd		vtStart	vtEnd
s1	{a1, a2, a3, a4}	01/01/1955	01/01/1965	{b1, b2}	01/01/1955	12/31/9999	{r1}	01/01/1965	12/31/9999
	{a1, a4, a5}	01/01/1965	12/31/9999						

Table 10.7 The AREA temporal typed table as a meta table about areas after the process of the RegionView_isb_1965.wrl file for the period of valid time from 01/01/1965 to the infinity

REFC	has_ ObjectID	has_3DObject			has_BoundingBox		
		vtValue	ValidTime		vtValue	ValidTime	
			vtStart	vtEnd		vtStart	vtEnd
a1	Parea1	tf6	01/01/1955	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.110025, 0, 31.620625, 42.942775, 0, 68.286375))	01/01/1955	12/31/9999
a2	Parea2	tf7	01/01/1955	01/01/1965	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(42.44075, 0, 31.620625, 78.52325, 0, 68.286375))	01/01/1955	01/01/1965
a3	Parea3	tf8	01/01/1955	01/01/1965	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(78.021075, 0, 31.620625, 103.604325, 0, 68.286375))	01/01/1955	01/01/1965
a4	Parea4	tf9	01/01/1955	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.10995, 0, 67.7843, 103.60445, 0, 92.7843))	01/01/1955	12/31/9999
a5	Parea5	tf12	01/01/1965	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(42.44085, 0, 31.620625, 103.60435, 0, 68.286375))	01/01/1965	12/31/9999

Table 10.7 (cont.)

REFC	has_AreaCode	belongs_to_AreaOwner			LIFESPAN	
		vtValue	ValidTime		lsStart	lsEnd
			vtStart	vtEnd		
a1	1	Jeff	01/01/1955	12/31/9999	01/01/1955	12/31/9999
a2	1	Jane	01/01/1955	01/01/1965	01/01/1955	01/01/1965
a3	1	John	01/01/1955	01/01/1965	01/01/1955	01/01/1965
a4	1	Julia	01/01/1955	12/31/9999	01/01/1955	12/31/9999
a5	1	Jane	01/01/1965	12/31/9999	01/01/1965	12/31/9999

Table 10.7 (cont.)

REFC	belongs_to_Region1Scene			is_categorized_by_AreaModel		
	vtValue	ValidTime		vtValue	ValidTime	
		vtStart	vtEnd		vtStart	vtEnd
a1	s1	01/01/1955	12/31/9999	am1	01/01/1955	12/31/9999
a2	s1	01/01/1955	01/01/1965	am1	01/01/1955	01/01/1965
a3	s1	01/01/1955	01/01/1965	am2	01/01/1955	01/01/1965
a4	s1	01/01/1955	12/31/9999	am1	01/01/1955	12/31/9999
a5	s1	01/01/1965	12/31/9999	am2	01/01/1965	12/31/9999

Table 10.8 The ROAD temporal typed table as a meta table about roads after the process of the RegionView_isb_1965.wrl file for the period of valid time from 01/01/1965 to the infinity

REFC	has_ ObjectID	has_3DObject			has_BoundingBox		
		vtValue	ValidTime		vtValue	ValidTime	
			vtStart	vtEnd		vtStart	vtEnd
r1	Pstreet1	tf13	01/01/1965	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.1973004, 0, 28.20886, 103.5171, 0, 32.03494))	01/01/1965	12/31/9999

Table 10.8 (cont.)

REFC	has_ RoadName	belongs_to_Region1Scene			is_categorized_by_RoadModel			LIFESPAN	
		vtValue	ValidTime		vtValue	ValidTime		lsStart	lsEnd
			vtStart	vtEnd		vtStart	vtEnd		
r1	Street1	s1	01/01/1965	12/31/9999	rm1	01/01/1965	12/31/9999	01/01/1965	12/31/9999

Table 10.9 A part of the REGION1SCENE temporal typed table as a meta table about regions after the process of the RegionView_isb_1985.wrl file for the period of valid time from 01/01/1985 to the infinity

REFC	contains_Area			contains_Building			contains_Road		
	vtValue	ValidTime		vtValue	ValidTime		vtValue	ValidTime	
		vtStart	vtEnd		vtStart	vtEnd		vtStart	vtEnd
s1	{a1, a2, a3, a4}	01/01/1955	01/01/1965	{b1, b2}	01/01/1955	01/01/1985	{r1}	01/01/1965	12/31/9999
	{a1, a4, a5}	01/01/1965	12/31/9999	{b1, b3}	01/01/1985	12/31/9999			

Table 10.10 The AREA temporal typed table as a meta table about areas after the process of the RegionView_isb_1985.wrl file for the period of valid time from 01/01/1985 to the infinity

REFC	has_ ObjectID	has_3DObject			has_BoundingBox		
		vtValue	ValidTime		vtValue	ValidTime	
			vtStart	vtEnd		vtStart	vtEnd
a1	Parea1	tf6	01/01/1955	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.110025, 0, 31.620625, 42.942775, 0, 68.286375))	01/01/1955	12/31/9999
a2	Parea2	tf7	01/01/1955	01/01/1965	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(42.44075, 0, 31.620625, 78.52325, 0, 68.286375))	01/01/1955	01/01/1965
a3	Parea3	tf8	01/01/1955	01/01/1965	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(78.021075, 0, 31.620625, 103.604325, 0, 68.286375))	01/01/1955	01/01/1965
a4	Parea4	tf9	01/01/1955	01/01/1985	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(172.10995, 0, 67.7843, 103.60445, 0, 92.7843))	01/01/1955	01/01/1985
		tf14	01/01/1985	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.109925, 0, 67.7843, 108.270675, 0, 92.7843))	01/01/1985	12/31/9999
a5	Parea5	tf12	01/01/1965	01/01/1985	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(42.44085, 0, 31.620625, 103.60435, 0, 68.286375))	01/01/1965	01/01/1985
		tf15	01/01/1985	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2),	01/01/1985	12/31/9999

					SDO_ORDINATE_ARRAY(42.4407, 0, 31.620625, 108.2707, 0, 68.286375))		

Table 10.11 The BUILDING temporal typed table as a meta table about buildings after the process of the RegionView_isb_1985.wrl file for the period of valid time from 01/01/1985 to the infinity

REFC	has_ ObjectID	has_3DObject			has_BoundingBox		
		vtValue	ValidTime		vtValue	ValidTime	
			vtStart	vtEnd		vtStart	vtEnd
b1	Pbuilding1	tf10	01/01/1955	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(19.54722, 0, 38.2631, 39.5539774, 16.5003, 48.2631))	01/01/1955	12/31/9999
b2	Pbuilding2	tf11	01/01/1955	01/01/1985	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(58.52348, 0, 37.9541, 71.18972, 13.00058, 56.1201))	01/01/1955	01/01/1985
b3	Pbuilding3	tf16	01/01/1985	12/31/9999	SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(32.27564, 0, 76.4509, 61.85716, 10.08416, 86.4509))	01/01/1985	12/31/9999

Table 10.11 (cont.)

REFC	has_BuildingName	belongs_to_BuildingOwner			LIFESPAN	
		vtValue	ValidTime		lsStart	lsEnd
			vtStart	vtEnd		
b1	Building1	Jeff	01/01/1955	12/31/9999	01/01/1955	12/31/9999
b2	Building2	Jane	01/01/1955	12/31/9999	01/01/1955	01/01/1985
b3	School	NULL	NULL	NULL	01/01/1985	12/31/9999

Table 10.11 (cont.)

REFC	belongs_to_Region1Scene			is_categorized_by_BuildingModel		
	vtValue	ValidTime		vtValue	ValidTime	
		vtStart	vtEnd		vtStart	vtEnd
b1	s1	01/01/1955	12/31/9999	bm2	01/01/1955	12/31/9999
b2	s1	01/01/1955	01/01/1985	bm1	01/01/1955	01/01/1985
b3	s1	01/01/1985	12/31/9999	bm1	01/01/1985	12/31/9999

Table 10.12 The AreaModel temporal typed table as a meta table about categories of areas after the process of the RegionView_isb_1985.wrl file for the period of valid time from 01/01/1985 to the infinity

REFC	has_PatternID	has_LandSort			categorizes_Area		
		vtValue	ValidTime		vtValue	ValidTime	
			vtStart	vtEnd		vtStart	vtEnd
am1	AreaModel1	NULL	NULL	NULL	{a1, a2, a4}	01/01/1955	12/31/1965
					{a1, a4}	01/01/1965	12/31/9999
am2	AreaModel2	NULL	NULL	NULL	{a3}	01/01/1955	01/01/1965
					{a5}	01/01/1965	12/31/9999

Table 10.13 The BuildingModel temporal typed table as a meta table about categories of buildings after the process of the RegionView_isb_1985.wrl file for the period of valid time from 01/01/1985 to the infinity

REFC	has_PatternID	is_for_UsePurpose			categorizes_Building		
		vtValue	ValidTime		vtValue	ValidTime	
			vtStart	vtEnd		vtStart	vtEnd
bm1	BuildingModel1	NULL	NULL	NULL	{b2}	01/01/1955	01/01/1985
					{b3}	01/01/1985	12/31/9999
bm2	BuildingModel2	NULL	NULL	NULL	{b1}	01/01/1955	12/31/9999

Table 10.14 The RoadModel temporal typed table as a meta table about categories of roads after the process of the RegionView_isb_1985.wrl file for the period of valid time from 01/01/1985 to the infinity

REFC	has_PatternID	is_for_VehicleKind			categorizes_Road		
		vtValue	ValidTime		vtValue	ValidTime	
			vtStart	vtEnd		vtStart	vtEnd
rm1	RoadModel1	NULL	NULL	NULL	{r1}	01/01/1965	12/31/9999

10.5 Summary

The temporal data population process from a 3D graphical data source with valid time has been presented with consideration on both graphical data and metadata parts of temporal 3D objects. The process can be automatically conducted using the database approach with the assumptions earlier stated in chapter 1. The preparation for the process is also discussed. After the successful execution of the process, the temporal object relational database for temporal 3D objects is available for any access in any non-temporal/temporal aspects at both graphical and semantic data levels.

Chapter 11

Querying 3D Objects with Valid Time

Once 3D graphical data and temporal metadata about temporal 3D objects and their scenes are available, the querying of temporal 3D objects can be performed by using the proposed temporal OR SQL language in chapter 8. In the proposed system, a wide range of queries are allowed to be related to 3D graphical data and/or non-explicit/explicit metadata with and without the temporal aspect. Defined in chapter 9, metadata about temporal 3D objects are separated into several meta tables corresponding to their individual semantic object types. Such an approach enables additional descriptive information on temporal 3D objects of the same object type, i.e. non-explicit metadata, to be appropriately included into a corresponding meta table. This also produces the extensibility of our 3D metadata handling towards any particular application domain that might deal with 3D objects with their other special non-presentable characteristics. In comparison with those supporting content-based search for individual 3D objects from an arbitrary collection/archive in [9, 34, 36] or for individual 3D objects in a scene in [35], our system can not only make queries about the existence of individual 3D objects in a scene but also enrich a wide range of queries in many aspects as follows.

Related to explicit metadata, the most interesting queries are those issued with spatial relationships among 3D objects in a scene. As mentioned earlier, our system provides a collection of 3D spatial functions in the Cartesian, right-handed, three-dimensional coordinate system based on the ISO standard [40]. Among these 3D spatial functions, 3D functions for measurement are distance3Dx, distance3Dy, distance3Dz, distance3D, height3D, width3D, and length3D. 3D functions for direction checking include left3D, on3D, above3D, and front3D. 3D functions for topological relationship checking consist of equal3D, disjoint3D, intersects3D, touches3D, crosses3D, within3D, contains3D, overlaps3D, and onBoundary3D. These 3D spatial functions are not available in any existing ORDBMS that implements the specification in [112] as operations on 3D or higher 3D spatial data ignore the Z dimension. To reduce the mathematics complexity of the checking on interiors, boundaries, and exteriors of 3D

objects, these proposed 3D spatial functions are performed on their bounding boxes. Moreover, we can use these 3D topological functions to answer queries about composition relationships between 3D objects. For instance, if an A object is composed of two B and C objects, the bounding box of A must contain the ones of B and C. In other words, the bounding boxes of B and C must be within the one of A. Thus, the two contains3D and within3D functions can be invoked to derive the composition of A. This support enlarges the range of queries in this work as compared to those works with a content-based search mechanism and the system in [12, 13] where only the checking of spatial relationships in direction (left, right, above, below, near, far) is supported.

For temporal requirements, it is claimed that the proposed system is the first one that takes into consideration the temporal aspect of 3D objects and scenes at both graphical and semantic data levels. In chapter 8, the ASELECT temporal selection operation, temporal operators [21] including ALWAYS, ANYTIME, NEXT, LAST, and UNTIL together with Allen's operators [100] including tBEFORE, tEQUAL, tMEETS, tOVERLAPS, tDURING, tSTARTS, and tFINISHES have been defined to express queries involving time. Nevertheless, direct access to time is possible with points in time and periods of time in a non-sequenced manner. This feature allows a non-restrictive temporal query formulation. The temporal support of this proposed system helps users capture not only the latest state of 3D objects and scenes but also others over the time.

In addition, non-temporal and temporal aggregations are able to be made over 3D objects and scenes with the aid of the proposed temporal object relational SQL language while unable in [4, 6, 8, 9, 12, 13, 26, 27, 39]. Such aggregations are useful for analysis and decision making support from a large collection of available 3D objects and scenes along time.

Furthermore, with the link established between graphical data and metadata of 3D objects and scenes, particular parts of a scene can be referred to and restored in VRML at a few various levels of fineness from object to scene. This facility makes our system different from the system in [12, 13] that allows the retrieval of entire scenes. Also, scene re-construction is permitted with any part of a scene for any period of valid time.

As an illustration, the querying of temporal 3D objects against temporal meta tables, graphical typed tables, and the T3DModel typed table is shown below.

11.1 Non-temporal Queries

11.1.1 Related to the Graphical Data Part of 3D Objects

(Q1). A query with aggregation related to the 3D graphical data content: How many objects were created from the given pattern: Shape(Appearance_t (Material_t (1, float3_t (0, 0, 0.823529), float3_t (0, 0, 0), 0.2, float3_t (0, 0, 0), 0), NULL, NULL), SYS.ANYDATA.convertObject (IndexedFaceSet (NULL, Coordinate (m_float3_t (float3_t (12.5, 0, 12.5), float3_t (12.5, 0, -12.5), float3_t (-12.5, 0, 12.5), float3_t (-12.5, 0, -12.5))), NULL, NULL, 'TRUE', m_integer_t(), 'FALSE', 'FALSE', m_integer_t (2, 0, 1, 3, -1), 0, m_integer_t(), 'TRUE', 'TRUE', m_integer_t()))))?

```
SELECT COUNT(a.REFC) INTO :AREA#
FROM AREA a, AREAMODEL ap
WHERE a.is_categorized_by_AreaModel = ap.REFC
AND ap.REFC = (SELECT REFC
        FROM T3DMODEL
        WHERE has_Gfx_Pattern.NodeType = 0
        AND has_Gfx_Pattern.NodeID.accessUROWID() = (SELECT REFC
                                                     FROM SHAPE_TAB s
                                                     WHERE isShapeEqual(VALUE(s), Shape(... )) = 'TRUE'));

SELECT COUNT(b.REFC) INTO :BUILDING#
FROM BUILDING b, BUILDINGMODEL bp
WHERE b.is_categorized_by_BuildingModel = bp.REFC
AND bp.REFC = (SELECT REFC
        FROM T3DMODEL
        WHERE has_Gfx_Pattern.NodeType = 0
        AND has_Gfx_Pattern.NodeID.accessUROWID() = (SELECT REFC
                                                     FROM SHAPE_TAB s
                                                     WHERE isShapeEqual(VALUE(s), Shape(... )) = 'TRUE'));

SELECT COUNT(r.REFC) INTO :ROAD#
FROM ROAD r, ROADMODEL rp
WHERE r.is_categorized_by_RoadModel = rp.REFC
AND rp.REFC = (SELECT REFC
        FROM T3DMODEL
        WHERE has_Gfx_Pattern.NodeType = 0
        AND has_Gfx_Pattern.NodeID.accessUROWID() = (SELECT REFC
                                                     FROM SHAPE_TAB s
                                                     WHERE isShapeEqual(VALUE(s), Shape(... )) = 'TRUE'));

OBJECT# = :AREA# + :BUILDING# + :ROAD#  = 3
```

11.1.2 Related to the Metadata Part of 3D Objects

(Q2). A non-temporal query: List all buildings higher than 12 meters.

```
SELECT has_ObjectID AS BUILDING_NAME, height3D(has_BoundingBox) AS HEIGHT
FROM BUILDING
WHERE height3D(has_BoundingBox) > 12;
```

BUILDING_NAME	HEIGHT
Pbuilding1	16.5003

(Q3). A non-temporal query: List all areas adjacent to a road.

```
SELECT a.has_ObjectID AS AREA_NAME, r.has_ObjectID AS ROAD_NAME
FROM AREA a, ROAD r
WHERE touches3D (a.has_BoundingBox, r.has_BoundingBox) = 'TRUE';
```

AREA_NAME	ROAD_NAME
Parea1	Pstreet1
Parea5	Pstreet1

(Q4). A non-temporal query: List all neighbors of the Parea1 area.

```
SELECT a2.has_ObjectID AS AREA_NAME
FROM AREA a1, AREA a2
WHERE a1.REFC != a2.REFC
AND a1.has_ObjectID = 'Parea1'
AND touches3D (a1.has_BoundingBox, a2.has_BoundingBox) = 'TRUE';
```

AREA_NAME
Parea4
Parea5

(Q5). A non-temporal query with aggregation: How many buildings are there on each area?

```
SELECT a.has_ObjectID AS AREA_NAME, COUNT(b.REFC) AS BUILDING#
FROM AREA a, BUILDING b
WHERE on3D(b.has_BoundingBox, a.has_BoundingBox) = 'TRUE'
GROUP BY a.has_ObjectID;
```

AREA_NAME	BUILDING#
Parea1	1
Parea4	1

(Q6). A non-temporal query with aggregation: List all areas where the highest buildings are located.

```
SELECT a.has_ObjectID AS AREA_NAME
FROM AREA a, BUILDING b
WHERE on3D(b.has_BoundingBox, a.has_BoundingBox) = 'TRUE'
AND height3D(b.has_BoundingBox) = (SELECT MAX (height3D (b.has_BoundingBox))
                                   FROM BUILDING);
```

AREA_NAME
Parea1

(Q7). A non-temporal query related to non-explicit metadata: List areas owned by Jane.

```
SELECT has_ObjectID
FROM AREA
WHERE belongs_to_AreaOwner = 'Jane';
```

has_ObjectID
Parea5

11.1.3 Related to both Graphical Data and Metadata Parts of 3D Objects

(Q8). List buildings in dark yellow of which the building owned by Jeff is on the left side.

```
SELECT b.has_ObjectID
FROM Building b, Building b2
WHERE b.has_3DObject IN (SELECT DISTINCT tf.REFC
                         FROM Transform_tab tf, TABLE(tf.children) c
                         WHERE c.NodeType = 0
                         AND GetUROWID(c.NodeID) = (SELECT s.REFC
                                                    FROM Shape_tab s
                                                    WHERE s.appearance.material.diffuseColor = float3_t(0.32549, 0.331373, 0.2313726))
AND b.REFC<>b2.REFC
AND b2.belongs_to_BuildingOwner = 'Jeff'
AND left3D(b2.has_BoundingBox, b.has_BoundingBox) = 'TRUE';
```

b.has_ObjectID
Pbuilding3

11.2 Temporal Queries

11.2.1 Related to the Graphical Data Part of 3D Objects

(Q9). List all dark yellow buildings some time in the past, present, or future.

```
SELECT b.has_ObjectID
FROM Building b
WHERE ANYTIME {b.has_3DObject IN (SELECT DISTINCT tf.REFC
                FROM Tarnsform_tab tf, TABLE(tf.children) c
                WHERE c.NodeType = 0
                AND GetUROWID(c.NodeID) = (SELECT s.REFC
                        FROM Shape_tab s
                        WHERE s.appearance.material.diffuseColor = float3_t(0.32549, 0.331373, 0.2313726))};
```

b.has_ObjectID
Pbuilding2
Pbuilding3

11.2.2 Related to the Metadata Part of 3D Objects

(Q10). A temporal query: List all buildings always higher than 12 meters.

```
SELECT has_ObjectID AS BUILDING_NAME, ASELECT{height3D(has_BoundingBox)} AS HEIGHT
FROM BUILDING
WHERE ALWAYS {height3D(has_BoundingBox) > 12};
```

BUILDING_NAME	HEIGHT		
	vtValue	vtStart	vtEnd
Pbuilding1	16.5003	01/01/1955	12/31/9999
Pbuilding2	13.00058	01/01/1955	01/01/1985

(Q11). A temporal query: List all areas adjacent to a road any time.

```
SELECT a.has_ObjectID AS AREA_NAME, r.has_ObjectID AS ROAD_NAME
FROM AREA a, ROAD r
WHERE ANYTIME {touches3D (a.has_BoundingBox, r.has_BoundingBox) = 'TRUE'};
```

AREA_NAME	ROAD_NAME
Parea1	Pstreet1
Parea5	Pstreet1

(Q12). A temporal query: List all neighbors of the Parea1 area any time.

```
SELECT a2.has_ObjectID AS AREA_NAME
FROM AREA a1, AREA a2
WHERE a1.REFC != a2.REFC
AND a1.has_ObjectID = 'Parea1'
AND ANYTIME {touches3D (a1.has_BoundingBox, a2.has_BoundingBox) = 'TRUE'};
```

AREA_NAME
Parea2
Parea4
Parea5

(Q13). A temporal query: List all buildings constructed before the Pbuilding3 building.

```
SELECT b1.has_ObjectID AS BUILDING_NAME
FROM BUILDING b1, BUILDING b2
WHERE b2.has_ObjectID = 'Pbuilding3'
AND b1.LIFESPAN.lsStart < b2.LIFESPAN.lsStart;
```

BUILDING_NAME
Pbuilding1
Building2

(Q14). A temporal query with aggregation: How many buildings have there been on each area from 01/01/1970 to 01/01/1980?

```
SELECT a.has_ObjectID AS AREA_NAME, COUNT(b.REFC) AS BUILDING#
FROM AREA a, BUILDING b
WHERE ALWAYS PERIOD ('01/01/1970', '01/01/1980') {on3D(b.has_BoundingBox, a.has_BoundingBox) = 'TRUE'}
GROUP BY a.has_ObjectID;
```

AREA_NAME	BUILDING#
Parea1	1
Parea5	1

(Q15). A temporal query with aggregation: List all areas where the highest buildings have been located from 01/01/1970 to 01/01/1980.

```
CREATE TABLE tmpMAX
NESTED TABLE HIGHEST STORE AS highest_nt
(NESTED TABLE ValidTime STORE AS ValidTime_nt)
AS SELECT tMAX {ASELECT PERIOD ('01/01/1970', '01/01/1980') {height3D (b.has_BoundingBox)}} AS HIGHEST
FROM BUILDING;

SELECT MAX(h.vtValue.AccessFloat()) INTO :highest
FROM tmpMAX t, TABLE(t.HIGHEST) h

SELECT a.has_ObjectID AS AREA_NAME
FROM AREA a, BUILDING b
WHERE ALWAYS PERIOD ('01/01/1970', '01/01/1980') {on3D(b.has_BoundingBox, a.has_BoundingBox) = 'TRUE'}
AND ALWAYS PERIOD ('01/01/1970', '01/01/1980') {height3D (b.has_BoundingBox) = :highest};
```

AREA_NAME
Parea1

(Q16). A temporal query related to non-explicit metadata: List areas owned by Jane from 1980 to 1990.

```
SELECT has_ObjectID
FROM AREA
WHERE ALWAYS PERIOD ('01/01/1980', '01/01/1990') {belongs_to_AreaOwner = 'Jane'};
```

has_ObjectID
Parea5

11.2.3 Related to both Graphical Data and Metadata Parts of 3D Objects

(Q17). A temporal query that returns VRML descriptions of 3D objects: View all objects that existed 30 years ago but not now.

```
CREATE TABLE AREA_VRML AS
SELECT VALUE(t).toVRML() AS ADescription
FROM TRANSFORM_TAB t
WHERE t.REFC IN (SELECT has_3DObject.vtValue
        FROM AREA
        WHERE has_3DObject.vtStart <= TO_DATE('01/01/1977', 'mm/dd/yyyy')
        AND tBEFORE (LIFESPAN, PERIOD (SYSDATE, '12/31/9999')) = 'TRUE');

CREATE TABLE BUILDING_VRML AS
SELECT VALUE(t).toVRML() AS BDescription
FROM TRANSFORM_TAB t
WHERE t.REFC IN (SELECT has_3DObject.vtValue
        FROM BUILDING
        WHERE has_3DObject.vtStart <= TO_DATE('01/01/1977', 'mm/dd/yyyy')
        AND tBEFORE (LIFESPAN, PERIOD (SYSDATE, '12/31/9999')) = 'TRUE');

CREATE TABLE ROAD_VRML AS
```

```
SELECT VALUE(t).toVRML() AS RDescription
FROM TRANSFORM_TAB t
WHERE t.REFC IN (SELECT has_3DObject.vtValue
        FROM ROAD
        WHERE has_3DObject.vtStart <= TO_DATE('01/01/1977', 'mm/dd/yyyy')
        AND tBEFORE (LIFESPAN, PERIOD (SYSDATE, '12/31/9999')) = 'TRUE');

Transform {
        center 0 0 0
        children [
                Shape {
                        appearance Appearance {
                                material Material {
                                        ambientIntensity 1
                                        diffuseColor .32549 .331373 .2313726
                                        emissiveColor 0 0 0
                                        shininess .2
                                        specularColor 0 0 0
                                        transparency 0
                                }
                                texture NULL
                                textureTransform NULL
                        }
                        geometry Box {
                                size 8 20 10
                        }
                }
        ]
        rotation 0 0 1 0
        scale 1.58328 .650029 1.8166
        scaleOrientation 0 0 1 0
        translation 64.8566 6.50029 47.0371
        bboxCenter 0 0 0
        bboxSize -1 -1 -1
}
```

Figure 11.1 The VRML description of the Pbuilding2 building that is a returned 3D object satisfying the query

(Q18). A temporal query that returns VRML descriptions of 3D objects: Form a scene of the Region1 region with all objects that have existed all the time from 01/01/1960 to 01/01/1970.

Figure 11.2 The 3D perspective view of the resulted *.wrl file with the addition of "#VRML V2.0 utf8" for the interpretation of a VRML browser

```
CREATE TABLE AREA_VRML AS
SELECT VALUE(t).toVRML() AS ADescription
FROM TRANSFORM_TAB t
WHERE t.REFC IN (SELECT has_3DObject.vtValue
        FROM AREA
        WHERE has_3DObject.vtStart <= TO_DATE('01/01/1960', 'mm/dd/yyyy')
        AND has_3DObject.vtEnd >= TO_DATE('01/01/1970', 'mm/dd/yyyy'));

CREATE TABLE BUILDING_VRML AS
SELECT VALUE(t).toVRML() AS BDescription
FROM TRANSFORM_TAB t
WHERE t.REFC IN (SELECT has_3DObject.vtValue
        FROM BUILDING
        WHERE has_3DObject.vtStart <= TO_DATE('01/01/1960', 'mm/dd/yyyy')
        AND has_3DObject.vtEnd >= TO_DATE('01/01/1970', 'mm/dd/yyyy'));

CREATE TABLE ROAD_VRML AS
SELECT VALUE(t).toVRML() AS RDescription
```

```
FROM TRANSFORM_TAB t
WHERE t.REFC IN (SELECT has_3DObject.vtValue
        FROM ROAD
        WHERE has_3DObject.vtStart <= TO_DATE('01/01/1960', 'mm/dd/yyyy')
        AND has_3DObject.vtEnd >= TO_DATE('01/01/1970', 'mm/dd/yyyy'));
```

(Q19). List buildings always in dark yellow of which the building some time owned by Jeff are now on the left side.

```
SELECT b.has_ObjectID
FROM Building b, Building b2
WHERE ALWAYS {b.has_3DObject IN (SELECT DISTINCT tf.REFC
                FROM Transform_tab tf, TABLE(tf.children) c
                WHERE c.NodeType = 0 AND
                GetUROWID(c.NodeID) = (SELECT s.REFC
                        FROM Shape_tab s
                        WHERE s.appearance.material.diffuseColor = float3_t(0.32549, 0.331373, 0.2313726))}
AND b.REFC<>b2.REFC
AND ANYTIME {b2.belongs_to_BuildingOwner = 'Jeff'}
AND left3D(b2.has_BoundingBox, b.has_BoundingBox) = 'TRUE';
```

b.has_ObjectID
Pbuilding3

11.3 Summary

From the querying of 3D objects with regard to valid time, it is found that this dissertation work has brought database technology, particularly OR database technology and temporal database technology, to the management of 3D objects and scenes along time. All 3D graphical data contents of 3D objects and scenes once stored with valid time and life spans in our system are able to be used for queries and retrieved in VRML for display with respect to valid time. All temporal metadata about 3D objects and scenes is connected to their graphical parts and queried for access to 3D objects and scenes at the semantic data level more conveniently than searching for 3D objects and scenes at the graphical data level with no temporal aspect. With the standardization of the SQL language, various front-end applications that need to have access to 3D objects and scenes handled in the proposed system and/or require metadata about 3D objects and scenes can be independently developed. These applications can receive inputs and give outputs from/to end-users via a suitable user-friendly interface while communicating with our system in a free query approach. Our system gets SQL statements as inputs, differently from the system in [12, 13] that builds a user-interface restrictively for query-by-example style queries. Besides, other facilities of a database system that our system inherits are probably helpful for front-end applications. Those facilities are concurrency control, backup and recovery control, and security with access control, etc. that are important for data reusability and sharability.

Chapter 12

The Proposed Temporal Object Relational Database System for 3D Objects

In this chapter, a temporal object relational database system is proposed for the handling of 3D objects in regard to valid time. The proposed system is developed on top of the temporal compatible object relational database system that has been introduced in chapter 8. It is more elaborated for dealing with 3D objects over the time.

12.1 The Proposed System Architecture

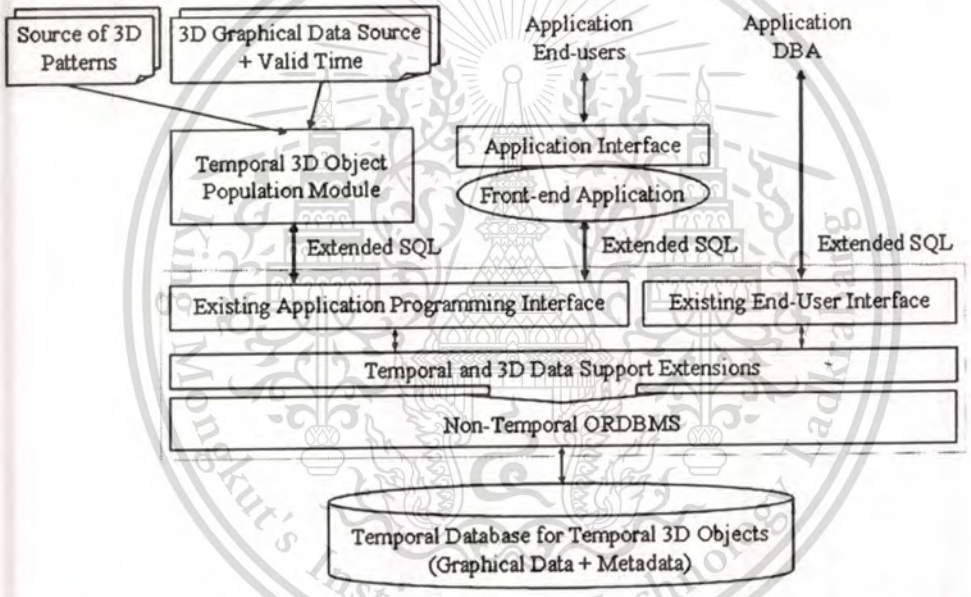


Figure 12.1 The Architecture of the Proposed Temporal Object Relational Database System for 3D Objects with regard to Valid Time

The architecture of this proposed temporal object relational database system for temporal 3D objects is described in Figure 12.1 where the temporal database for temporal 3D objects detailed in chapter 9 is based on the object relational data model. The system generates an environment for the managing of temporal 3D objects and their scenes populated from 3D graphical data sources with respect to valid time. The details of the system architecture are explained as follows.

- *Non-temporal ORDBMS*: as mentioned previously, the proposed system is developed on top of a non-temporal ORDBMS that allows the extensibility of abstract data types,

user-defined methods, and optimization methods as well as the supply of opaque data types. Such an ORDBMS can be Oracle 10g/11g with data cartridges, Informix with DataBlades, or Universal DB2 with Extenders. In this concrete implementation, the employed non-temporal ORDBMS is Oracle 10g/11g. Nevertheless, our work is not product-specific to the underlying ORDBMS. It can be portal to other ORDBMSs.

- ***Temporal and 3D data support extensions***: In the development of the proposed system, the temporal and 3D data support extensions are constructed as an add-on easily and directly integrated into the employed non-temporal ORDBMS. The extensions simply include two modules, one for 3D data support and the other for temporal data support. Each module contains our user-defined types and methods. These modules will be detailed in the next subsections 12.2 and 12.3. Thus, the support of our system for temporal 3D objects in particular and for temporal data in general can be practically reached by users (application programs and end-users).

- ***Temporal database for temporal 3D objects***: A temporal database for temporal 3D objects handled in the proposed system has been defined in chapter 9. It is a temporal object relational database composed of an object relational graphical database and a temporal object relational meta database for the graphical data and metadata parts of temporal 3D objects, respectively. The temporal aspect of 3D objects and metadata surrounding them is valid time captured at both object and attribute levels.

- ***Front-end applications and their database administrators (DBAs)***: The users of this proposed system are front-end application programs and their database administrators (DBAs) who can interactively have access to temporal 3D objects and their scenes as well as metadata surrounding them. By using the proposed temporal object relational SQL language in chapter 8, these users can non-restrictively consume the temporal object relational database built for temporal 3D objects and their scenes handled by the proposed system in the SQL standard manner.

- ***Application programming interfaces and end-user interfaces***: All existing application programming interfaces and end-user interfaces of the employed non-temporal ORDBMS are included in our system. Therefore, all other facilities supplied by the employed non-temporal ORDBMS can be made use of as well. These facilities are concurrency control, backup and recovery control, security with access control, and so

on that are important for data reusability and sharability. With the reuse of well-known application programming interfaces from the employed ORDBMS, front-end applications can be conveniently developed on top of this proposed system by means of any programming language that is supported by the employed non-temporal ORDBMS.

- ***Temporal 3D object population module***: For the temporal data population of the temporal object relational database from 3D graphical data sources, the temporal 3D object population module is built as an external module of this system on top of the temporal compatible object relational database system proposed in chapter 8. All processes defined for temporal data population in chapter 10 are implemented as the functions of this module. Indeed, this module takes as its inputs the sources of 3D patterns from which 3D objects and their scenes are generated as well as the input 3D graphical data sources and their valid time. The output of this module is a temporal object relational database for temporal 3D objects and their scenes. This module is allowed to be accessed by graphics designers, front-end application developers, and domain experts who are involved in making available a temporal object relational database for temporal 3D objects and their scene. A graphical user interface developed along with the module is displayed in Appendix B.

- ***Extended SQL***: The key component of this proposed system is the temporal object relational SQL language defined in chapter 8. The language is the extended object relational SQL language for temporal and 3D data supports. The language can be used in a temporal transparency environment in the SQL standard manner like the standard object relational SQL language. Both application programs and end-users can employ this language for access to temporal object relational databases handled by our system.

The rest of this chapter will be dedicated to the SQL extension modules which are the key components of the proposed system. These modules are the results of our system implementation, which is carried out by using the Oracle 10g/11g ORDBMS.

12.2 The SQL Extension Module for 3D Data Support

Based on the assumptions of this research, the SQL extension module for 3D data support is associated to the VRML language, the standard 3D object description language. This module is part of the integrated extension implementing the proposed

system, including a collection of 3D graphical data types corresponding to VRML field types and node types and a collection of 3D spatial methods from the standard specification in [40].

12.2.1 3D Graphical Data Types

In chapter 9, the 3D graphical data types corresponding to VRML field types and node types have already been introduced for the definition of the object relational graphical database of temporal 3D objects and their scenes. So, they are not repeated in this subsection. Instead, we pay attention to the specification of their member functions, namely toVRML() and compBBox(), which are mentioned elsewhere.

(a) toVRML()

- Input: an instance of the structured type corresponding to any node type defined in the VRML language.
- Output: a text that contains the VRML description of a node of the node type whose state is formed from the attribute values of the input instance of the corresponding structured type.
- Description: this function aims at the transformation of a graphical element stored in the object relational graphical database into a graphical element formatted in the VRML language that is able to be browsed with any VRML 3D graphics browser.

(b) compBBox()

- Input: an instance of the structured type corresponding to a geometric node type (Box, Cone, Cylinder, ElevationGrid, IndexedFaceSet, IndexedLineSet, PointSet, and Sphere) or a visible node type (Shape, Transform, and Group) whose nodes can refer to one or many nodes of a geometric node type defined in the VRML language. Such an instance is a graphical description of some 3D object or some component (part) of a 3D object.
- Output: the minimum bounding box that is represented by two coordinates, i.e. two points, Pmin = (Xmin, Ymin, Zmin) and Pmax = (Xmax, Ymax, Zmax). In our implementation using Oracle 10g/11g, the spatial data type SDO_GEOMETRY is used to wrap these two coordinates into one unit as an output of this member

function. Thus, the representation of a minimum bounding box with two coordinates by using an instance of the spatial data type SDO_GEOMETRY is given in Figure 12.2 as follows.

```
SDO_GEOMETRY (
        3005                                    -- SDO_GTYPE
        , NULL                                  -- SDO_SRID
        , NULL                                  -- SDO_POINT
        , SDO_ELEM_INFO (1, 1, 2)               -- SDO_ELEM_INFO
        , SDO_ORDINATES (                       -- SDO_ORDINATES
                Xmin, Ymin, Zmin,
                Xmax, Ymax, Zmax
        )
)
```

Figure 12.2 The Representation of a Minimum 3D Bounding Box using the Spatial Data Type SDO_GEOMETRY

- Description: The main challenge of this function is to determine the values of Xmin, Ymin, Zmin, Xmax, Ymax, Zmax according to the state of the method-invoking instance of the structured type corresponding to an instance of the aforementioned node type. Other node types including Extrusion, LOD, and Switch have not yet been supported.

  (i) Box in Figure 12.3

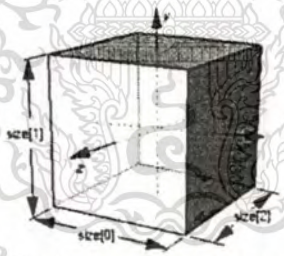


Figure 12.3 The Graphical Representation of a Node of the Box Node Type

Xmin = -size[0]/2; Ymin = -size[1]/2; Zmin = -size[2]/2;
Xmax = size[0]/2; Ymax = size[1]/2; Zmax = size[2]/2

(ii) Cone in Figure 12.4

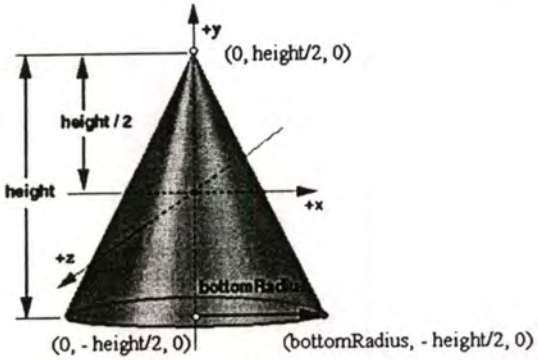


Figure 12.4 The Graphical Representation of a Node of the Cone Node Type

Xmin = -bottomRadius; Ymin = -height/2; Zmin = -bottomRadius;
Xmax = bottomRadius; Ymax = height/2; Zmax = bottomRadius

(iii) Cylinder in Figure 12.5

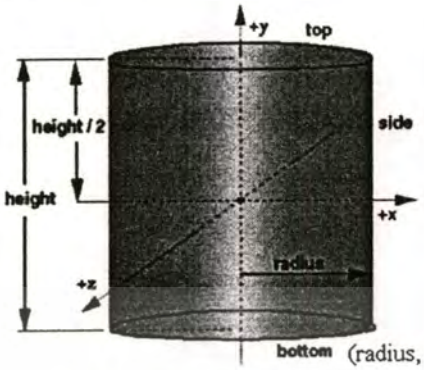


Figure 12.5 The Graphical Representation of a Node of the Cylinder Node Type

Xmin = -radius; Ymin = -height/2; Zmin = -radius;
Xmax = radius; Ymax = height/2; Zmax = radius

(iv) ElevationGrid in Figure 12.6

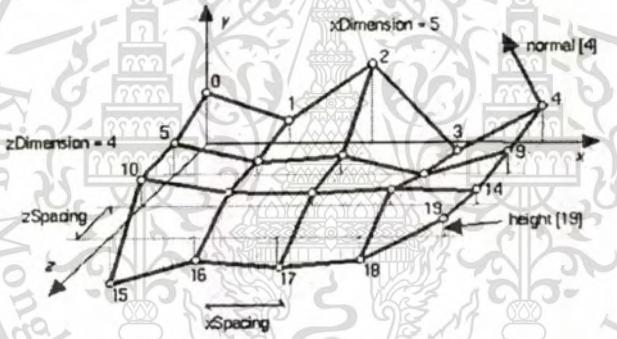


Figure 12.6 The Graphical Representation of a Node of the ElevationGrid Node Type

Let P[i, j] be a vertex (coordinate) on the grid where i >= 0 and i < xDimension, j >= 0 and j > zDimension. The x, y, z values for each P[i, j] are determined as follows.

P[i, j].x = xSpacing*i;
P[i, j].y = height[i+j*xDimension];
P[i, j].z = zSpacing*j

Let Max() and min() be the two functions that return the maximum and minimum values, respectively, from an input collection of values. The two minimum and maximum coordinates of the minimum 3D bounding box of a 3D object or part of a 3D object graphically represented by a node of the ElevationGrid node type are computed below.

Xmin = 0; Ymin = min(P[i, j].y); Zmin = 0;
Xmax = xSpacing*xDimension; Ymax = Max(P[i, j].y); Zmax = zSpacing*zDimension

(v) IndexedFaceSet in Figure 12.7

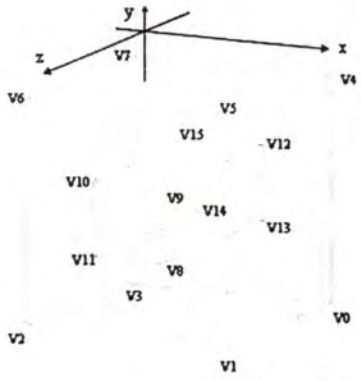


Figure 12.7 The Graphical Representation of a Node of the IndexedFaceSet Node Type

Let Vi be a vertex (coordinate) of a 3D object or part of a 3D object graphically represented by a node of the IndexedFaceSet node type where i >= 1 and i <= n that is a positive integer number. Each Vi is identified by Vi.x, Vi.y, and Vi.z along the x, y, and z axes, respectively. The minimum 3D bounding box of such a 3D object or part of a 3D object is computed as follows.

Xmin = min(Vi.x); Ymin = min(Vi.y); Zmin = min(Vi.z);
Xmax = Max(Vi.x); Ymax = Max(Vi.y); Zmax = Max(Vi.z)

(vi) IndexedLineSet in Figure 12.8

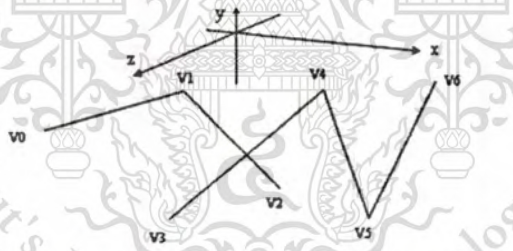


Figure 12.8 The Graphical Representation of a Node of the IndexedLineSet Node Type

The computation of the minimum 3D bounding box of a 3D object or part of a 3D object graphically represented by a node of the IndexedLineSet node type is similar to the one of the IndexedFaceSet node type. Let Vi be a vertex (coordinate) of a 3D object or part of a 3D object where i >= 1 and i <= n that is a positive integer number. Each Vi is identified by Vi.x, Vi.y, and Vi.z along the x, y, and z axes, respectively. The minimum 3D bounding box of a 3D object or part of a 3D object is computed as follows.

Xmin = min(Vi.x); Ymin = min(Vi.y); Zmin = min(Vi.z);
Xmax = Max(Vi.x); Ymax = Max(Vi.y); Zmax = Max(Vi.z)

(vii) PointSet in Figure 12.9

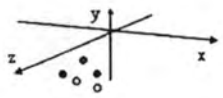


Figure 12.9 The Graphical Representation of a Node of the PointSet Node Type

The computation of the minimum bounding box of a 3D object or part of a 3D object graphically represented by a node of the PointSet node type is similar to the one of the IndexedFaceSet node type. Let Vi be a vertex (coordinate) of a 3D object or part of a 3D object where i >= 1 and i <= n that is a positive integer number. Each Vi is identified by Vi.x, Vi.y, and Vi.z along the x, y, and z axes, respectively. The minimum bounding box of a 3D object or part of a 3D object is computed as follows.

Xmin = min(Vi.x); Ymin = min(Vi.y); Zmin = min(Vi.z);
Xmax = Max(Vi.x); Ymax = Max(Vi.y); Zmax = Max(Vi.z)

(viii) Sphere in Figure 12.10

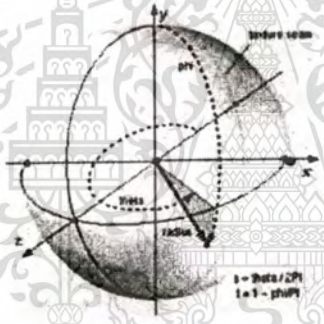


Figure 12.10 The Graphical Representation of a Node of the Sphere Node Type

Xmin = -radius; Ymin = -radius; Zmin = -radius;
Xmax = radius; Ymax = radius; Zmax = radius

(ix) Shape

A 3D object or part of a 3D object is graphically represented by a node of the Shape node type in which the geometry of the object is defined by a geometric node of the Box, Cone, Cylinder, ElevationGrid, Extrusion, IndexedFaceSet, IndexedLineSet, PointSet, or Sphere node type and the appearance of the 3D object or part of the 3D object is defined by a node of the Appearance node type. Therefore, the minimum bounding box of the 3D object or part of the 3D object graphically represented by a node of the Shape node type is in fact the minimum bounding box calculated for the geometric node included in the Shape node. The minimum bounding box associated with a geometric node has been obtained above in (i..viii).

(x) Group in Figure 12.11

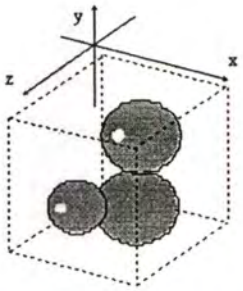


Figure 12.11 The Graphical Representation of a Node of the Group Node Type

A node of the Group node type contains nodes without introducing any new transformation. In other words, the Group node lets us treat a set of nodes as a single entity. Therefore, the minimum bounding box of a 3D object or part of a 3D object graphically represented by a node of the Group node type is the one large enough to enclose the union of all minimum bounding boxes associated with nodes included in the Group node that is being considered. Let BBoxi for i >= 1 be the minimum bounding box associated with a node included in the Group node in the field of the MFNode field type. The minimum bounding box of the 3D object or part of the 3D object graphically represented by a node of the Group node type is calculated as follows.

```
Xmin = min(BBoxi.Xmin); Ymin = min(BBoxi.Ymin); Zmin = min(BBoxi.Zmin);
Xmax = Max(BBoxi.Xmax); Ymax = Max(BBoxi.Ymax); Zmax = Max(BBoxi.Zmax)
```

(xi) Transform in Figure 12.12

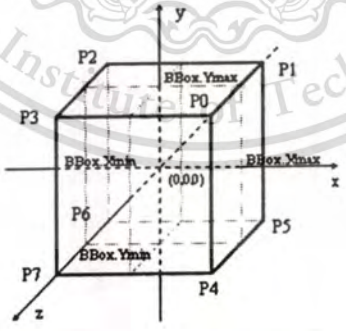


Figure 12.12 The Graphical Representation of a Node of the Transform Node Type

Similar to the Group node type, the Transform node type allows a node of this node type to be composed of a collection of nodes of other node types as a single entity to graphically represent some 3D object or part of some 3D object. An included node will be affected by the transformation defined by means of the values of the translation,

rotation, scale, scaleOrientation, and center fields that form a new coordinate system associated with a Transform node. We consider the transformation applied to a 3D point P included in a Transform node shown in Figure 12.13. That is, P is transformed into P' where P' = T x C x R x SR x S x –SR x –C x P and T, C, R, SR, and S are transformation matrices defined in the following Figures 12.14..12.22.

```
Transform {
        center          C
        rotation        R
        scale           S
        scaleOrientation SR
        translation     T
        children        [...P..]
}
```

Figure 12.13 A Transformation Node

$$T = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 12.14 The Translation Matrix T

$$C = \begin{bmatrix} 1 & 0 & 0 & cx \\ 0 & 1 & 0 & cy \\ 0 & 0 & 1 & cz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 12.15 The Center Matrix C

$$Rx(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 12.16 The Rotation Matrix Rx or the ScaleOrientation Matrix SRx for an Angle Counterclockwise along the X Axis

$$R\text{-}x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\theta) & -\sin(-\theta) & 0 \\ 0 & \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 12.17 The Rotation Matrix R-x or the ScaleOrientation Matrix SR-x for an Angle Anti-counterclockwise along the X Axis

$$Ry(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 12.18 The Rotation Matrix Ry or the ScaleOrientation Matrix SRy for an Angle Counterclockwise along the Y Axis

$$R\text{-}y(\theta) = \begin{bmatrix} \cos(-\theta) & 0 & \sin(-\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\theta) & 0 & \cos(-\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 12.19 The Rotation Matrix R-y or the ScaleOrientation Matrix SR-y for an Angle Anti-counterclockwise along the Y Axis

$$Rz(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 12.20 The Rotation Matrix Rz or the ScaleOrientation Matrix SRz for an Angle Counterclockwise along the Z Axis

$$R\text{-}z(\theta) = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) & 0 & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 12.21 The Rotation Matrix R-z or the ScaleOrientation Matrix SR-z for an Angle Anti-counterclockwise along the Z Axis

$$S = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 12.22 The Scale Matrix S

Using the transformation mechanism above, the minimum bounding box of a composite object or composite part of an object graphically represented by a node of the Transform node type is calculated as follows.

Firstly, let BBox be the minimum bounding box stemming from all nodes included in the field of the MFNode field type of the Transform node. The computation of BBox is the same as the one for a node of the Group node type.

Secondly, let P0, P1, P2, P3, P4, P5, P6, and P7 be the eight coordinates of BBox.

P0 = (BBox.Xmax, BBox.Ymax, BBox.Zmax);
P1 = (BBox.Xmax, BBox.Ymax, BBox.Zmin);
P2 = (BBox.Xmin, BBox.Ymax, BBox.Zmin);
P3 = (BBox.Xmin, BBox.Ymax, BBox.Zmax);
P4 = (BBox.Xmax, BBox.Ymin, BBox.Zmax);
P5 = (BBox.Xmax, BBox.Ymin, BBox.Zmin);
P6 = (BBox.Xmin, BBox.Ymin, BBox.Zmin);
P7 = (BBox.Xmin, BBox.Ymin, BBox.Zmax)

Thirdly, the transformation of the Transform node that is being considered made by the transformation matrices above is applied to each individual point P0..P7 of BBox to obtain P0*..P7*, respectively.

$$Pi^* = T \times C \times R \times SR \times S \times -SR \times -C \times Pi \text{ for } i = 0..7$$

Fourthly, the minimum bounding box of BBox after the transformation is the minimum bounding box of a composite object or part of a composite object graphically represented by the Transform node that is being considered is determined as follows.

Xmin = min(Pi*.X); Ymin = min(Pi*.Y); Zmin = min(Pi*.Z);
Xmax = Max(Pi*.X); Ymax = Max(Pi*.Y); Zmax = Max(Pi*.Z)

12.2.2 3D Spatial Methods

A collection of 3D spatial methods supplied within our system is classified into topological methods, directional methods, and metric methods. These methods are used to determine spatial relationships among 3D objects in a 3D scene according to their positions. In order to diminish greatly the complexity of mathematics involved in the computation of the exteriors, interiors, and boundaries of 3D objects in these 3D spatial methods, the minimum bounding boxes of these 3D objects are used as inputs of these 3D spatial methods. The use of the minimum bounding boxes has a little impact on the precision of the calculation. Indeed, the difference is acceptable to let us know relatively about the spatial relationships among 3D objects in a 3D scene.

12.2.2.1 Topological Methods

Topological methods are equal3D, disjoint3D, intersects3D, touches3D, crosses3D, within3D, contains3D, overlaps3D, and onBoundary3D. These methods are based on the checking of the exteriors, interiors, and boundaries of the two input 3D objects as

shown in Figure 12.23. The calculation of the interior, exterior, and boundary of an actual 3D object is of very high complexity. So, the minimum bounding boxes of 3D objects are used in general. These minimum bounding boxes are represented by two coordinates (points) as the minimum point (Xmin, Ymin, Zmin) and the maximum point (Xmax, Ymax, Zmax). They are achieved for each individual 3D object after the temporal data population process from an input 3D graphical data source.

intersection [boundary (A), boundary (B)]	intersection [boundary (A), interior (B)]	intersection [boundary (A), exterior (B)]
intersection [interior (A), boundary (B)]	intersection [interior (A), interior (B)]	intersection [interior (A), exterior (B)]
intersection [exterior(A), boundary(B)]	intersection [exterior (A), interior (B)]	intersection [exterior (A), exterior (B)]

Figure 12.23 The 9 Intersection Matrix for Egenhofer's Operators [40]

The following shows how simple the checking of spatial relationships between two minimum bounding boxes of the two input 3D objects is in comparison with between the two input 3D objects. The input of each 3D spatial function is the two minimum bounding boxes of the two corresponding 3D objects. The output of each 3D spatial function is 'TRUE' or 'FALSE'.

(i) equal3D() in Figure 12.24

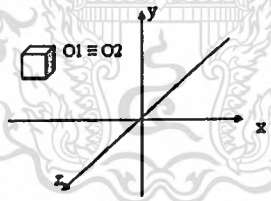


Figure 12.24 O1 is equal to O2.

equal3D(O1, O2) = 'TRUE'
iff bbox1.Xmin = bbox2.Xmin and bbox1.Xmax = bbox2.Xmax
and bbox1.Ymin = bbox2.Ymin and bbox1.Ymax = bbox2.Ymax
and bbox1.Zmin = bbox2.Zmin and bbox1.Zmax = bbox2.Zmax

(ii) disjoint3D() in Figure 12.25

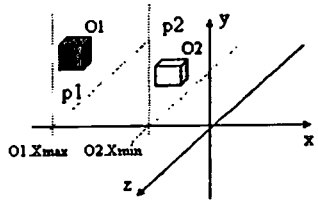


Figure 12.25 O1 is disjoint from O2.

disjoint3D(O1, O2) = 'TRUE'
iff bbox1.Xmax < bbox2.Xmin or bbox2.Xmax < bbox1.Xmin
or bbox1.Ymax < bbox2.Ymin or bbox2.Ymax < bbox1.Ymin

or bbox1.Zmax < bbox2.Zmin or bbox2.Zmax < bbox1.Zmin

(iii) intersects3D() in Figure 12.26

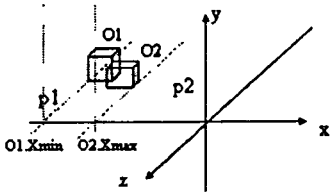


Figure 12.26 O1 intersects O2.

intersects3D(O1, O2) = NOT disjoint3D(O1, O2)

(iv) touches3D() in Figure 12.27

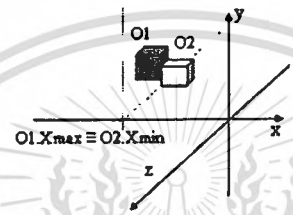


Figure 12.27 O1 touches O2.

touches3D(O1, O2) = 'TRUE'
iff ((bbox1.Xmax = bbox2.Xmin OR bbox2.Xmax = bbox1.Xmin) AND
((bbox1.Ymin <= bbox2.Ymin AND bbox2.Ymin <= bbox1.Ymax) OR
(bbox1.Ymin <= bbox2.Ymax AND bbox2.Ymax <= bbox1.Ymax)) AND
((bbox1.Zmin <= bbox2.Zmin AND bbox2.Zmin <= bbox1.Zmax) OR
(bbox1.Zmin <= bbox2..Zmax AND bbox2.Zmax <= bbox1.Zmax)))
OR ((bbox1.Ymax = bbox2.Ymin OR bbox2.Ymax = bbox1.Ymin) AND
((bbox1.Xmin <= bbox2.Xmin AND bbox2.Xmin <= bbox1.Xmax) OR
(bbox1.Xmin <= bbox2.Xmax AND bbox2.Xmax <= bbox1.Xmax)) AND
((bbox1.Zmin <= bbox2.Zmin AND bbox2.Zmin <= bbox1.Zmax) OR
(bbox1.Zmin <= bbox2.Zmax AND bbox2.Zmax <= bbox1.Zmax)))
OR ((bbox1.Zmax = bbox2.Zmin OR bbox2.Zmax = bbox1.Zmin) AND
((bbox1.Xmin <= bbox2.Xmin AND bbox2.Xmin <= bbox1.Xmax) OR
(bbox1.Xmin <= bbox2.Xmax AND bbox2.Xmax <= bbox1.Xmax)) AND
((bbox1.Ymin <= bbox2.Ymin AND bbox2.Ymin <= bbox1.Ymax) OR
(bbox1.Ymin <= bbox2.Ymax AND bbox2.Ymax <= bbox1.Ymax)))

(v) within3D() in Figure 12.28

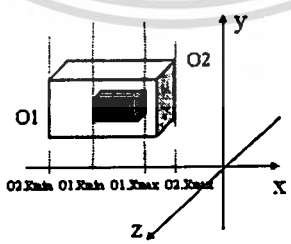


Figure 12.28 O1 is within O2.

within3D(O1, O2) = 'TRUE'
iff bbox2.Xmin <= bbox1.Xmin and bbox1.Xmax <= bbox2.Xmax
and bbox2.Ymin <= bbox1.Ymin and bbox1.Ymax <= bbox2.Ymax
and bbox2.Zmin <= bbox1.Zmin and bbox1.Zmax <= bbox2.Zmax

(vi) contains3D()

contains3D(O1, O2) = within3D(O2, O1)

(vii) overlaps3D()

overlaps3D(O1, O2) = intersects3D(O1, O2)

12.2.2.2 Directional Methods

These directional methods are boolean functions to check the spatial relationships between two objects based on the direction of the base axis. The notions of "left" and "right" are determined along the X axis; the notions of "above" and "below" along the Y axis; and the notions of "front" and "behind" along the Z axis. In addition, the notion of "on" is defined as a combination of the two methods touches3D() and above3D(). Similar to the topological methods, these directional methods also take as an input the two minimum bounding boxes of the two corresponding 3D objects. The output of each directional method is also 'TRUE' or 'FALSE'.

(i) left3D() in Figure 12.29

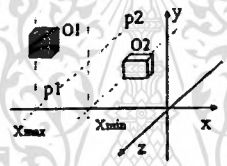


Figure 12.29 O1 is on the left side of O2.

left3D(O1, O2) = 'TRUE' iff bbox1.Xmax <= bbox2.Xmin

(ii) on3D() in Figure 12.30

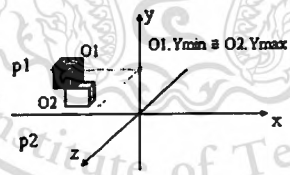


Figure 12.30 O1 is on O2.

on3D(O1, O2) = 'TRUE'
iff ((bbox1.Ymax = bbox2.Ymin OR bbox2.Ymax = bbox1.Ymin) AND
((bbox1.Xmin <= bbox2.Xmin AND bbox2.Xmin <= bbox1.Xmax) OR
(bbox1.Xmin <= bbox2.Xmax AND bbox2.Xmax <= bbox1.Xmax)) AND
((bbox1.Zmin <= bbox2.Zmin AND bbox2.Zmin <= bbox1.Zmax) OR
(bbox1.Zmin <= bbox2..Zmax AND bbox2.Zmax <= bbox1.Zmax)))

(iii) above3D() in Figure 12.31

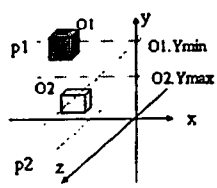


Figure 12.31 O1 is above O2.

above3D(O1, O2) = 'TRUE' iff bbox1.Ymin >= bbox2.Ymax

(iv) front3D() in Figure 12.32

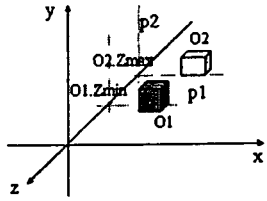


Figure 12.32 O1 is in front of O2.

front3D(O1, O2) = 'TRUE' iff bbox1.Zmin >= bbox2.Zmax

12.2.2.3 Metric Methods

Different from the two previous groups, this group of metric methods height3D(), width3D(), and length3D() approximately measures the height, width, and length of a 3D object through its minimum bounding box, respectively. Also, the distances between two 3D objects along the X, Y, and Z axis with the distance3Dx(), distance3Dy(), and distance3Dz() methods, respectively, and the shortest distance between two 3D objects in the 3D space with the distance3D() method are computed. The inputs of each of these metric methods are the minimum bounding boxes of the involved 3D objects. The output is some value of some measurement unit. The measurement unit used in the VRML language is meter.

(i) height3D() in Figure 12.33

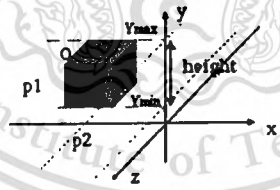


Figure 12.33 The Height of a 3D Object O

heigth3D(O) = bbox.Ymax – bbox.Ymin

(ii) width3D() in Figure 12.34

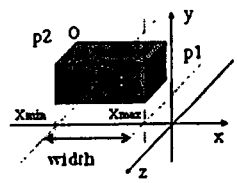


Figure 12.34 The Width of a 3D Object O

width3D(O) = bbox.Xmax – bbox.Xmin

(iii) length3D() in Figure 12.35

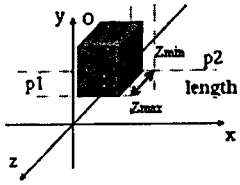


Figure 12.35 The Length of a 3D Object O

length3D(O) = bbox.Zmax – bbox.Zmin

(iv) distance3D()

distance3D(O1, O2) = the shortest Euclidean distance between any two points P1 and P2 where (P1 belongs to O1 and P2 belongs to O2) or (P1 belongs to O2 and P2 belongs to O1).

(v) distance3Dx()

distance3Dx(O1, O2) = Max{bbox2.Xmin – bbox1.Xmax, bbox1.Xmin – bbox2.Xmax, 0}

(vi) distance3Dy()

distance3Dy(O1, O2) = Max{bbox2.Ymin – bbox1.Ymax, bbox1.Ymin – bbox2.Ymax, 0}

(vii) distance3Dz()

distance3Dz(O1, O2) = Max{bbox2.Zmin – bbox1.Zmax, bbox1.Zmin – bbox2.Zmax, 0}

12.3 The SQL Extension Module for Temporal Data Support

For temporal data support in general, this proposed SQL extension module includes a collection of temporal data types and a collection of temporal methods. These data types and methods are used for the transformation of temporal SQL statements into conventional SQL statements temporally enhanced with our temporal data types and methods. Wrapped in one of our appropriate translation stored-procedures, a temporal SQL statement is directly submitted to the employed ORDBMS via an existing interface. Furthermore, the database users can take advantage of this SQL extension module for non-sequenced cases with non-restrictiveness. It is because these temporal data types and methods can be non-restrictively used in the same way as built-in and DBMS-supplied data types and methods of the employed ORDBMS.

12.3.1 Temporal Data Types

Corresponding to a period of time, a set of periods of time, a temporal composition, and an attribute history defined in chapter 8, four temporal data types VTPERIOD, VTPERIODS, ANYTEMPORALVALUE, and ANYTEMPORALDATA are defined. The VTPERIODS data type is a collection type of the VTPERIOD data type and

ANYTEMPORALDATA of ANYTEMPORALVALUE. The SYS.ANYDATA opaque data type provided by the Oracle 10g/11g ORDBMS is used as the data type of the time dependent component of a temporal composition so that our temporal support can be given to data of any data type. The built-in data type DATE is used as the data type of instants (points of time) on the time axis. Therefore, the DAY granularity is supported as of this moment. Nevertheless, a large number of built-in datetime related methods provided by Oracle 10g/11g can be applied to deal with other granularities such as YEAR, MONTH, HOUR, MINUTE, etc. The specifications of these four temporal data types are shown as follows.

(a) The VTPERIOD data type

```
CREATE OR REPLACE TYPE VTPERIOD AS OBJECT (
    vtStart     DATE,
    vtEnd       DATE,
    MEMBER FUNCTION IsWellFormed RETURN NUMBER,
    MEMBER FUNCTION IsCurrent RETURN NUMBER, --0 = TRUE if the interval contains "NOW"
    MEMBER FUNCTION GetDuration (granularity IN CHAR) RETURN NUMBER
);
/
```

(b) The VTPERIODS data type

```
CREATE OR REPLACE TYPE VTPERIODS AS TABLE OF VTPERIOD;
/
```

(c) The ANYTEMPORALVALUE data type

```
CREATE OR REPLACE TYPE ANYTEMPORALVALUE AS OBJECT (
    vtValue             SYS.ANYDATA,
    ValidTime           VTPERIODS,
    MEMBER FUNCTION IsWellFormed RETURN NUMBER,
    MEMBER FUNCTION IsCurrentData RETURN NUMBER --0 = TRUE if its valid time contains "NOW"
);
/
```

(d) The ANYTEMPORALDATA data type

```
CREATE OR REPLACE TYPE ANYTEMPORALDATA AS TABLE OF ANYTEMPORALVALUE;
/
```

12.3.2 Temporal Methods

The temporal methods of this module are classified into the following five groups: basic methods, sequenced methods, temporal aggregate functions, translation stored-procedures, and external functions.

12.3.2.1 Basic Methods

The group of basic methods consists of LifeSpan(), tValue(), tAnyData(), and ValidTime() corresponding to the ones introduced in subsection 8.2.

```
tBEFORE (period1, period2)          11111 22222          TRUE

tEQUAL (period1, period2)           11111
                                    22222                TRUE

tMEETS (period1, period2)           1111122222           TRUE

tOVERLAPS (period1, period2)        11111
                                      22222              TRUE

tDURING (period1, period2)            11111
                                    22222222             TRUE

tSTARTS (period1, period2)          11111
                                    22222222             TRUE

tFINISHES (period1, period2)          11111
                                    22222222             TRUE
```

Figure 12.36 Conventional Allen's Operators

```
tBEFORE (periods1, periods2)        111  11111 22222  22      TRUE

tEQUAL (periods1, periods2)         11111  111               TRUE
                                    22222  222

tMEETS (periods1, periods2)         111  1111122222 22       TRUE

tOVERLAPS (periods1, periods2)      11111  1111              TRUE
                                      22222       222

tDURING (periods1, periods2)          11111  11    11        TRUE
                                    22222222 222   22

tSTARTS (periods1, periods2)        11111  1111              TRUE
                                    22222222       222

tFINISHES (periods1, periods2)      111        11111         TRUE
                                      2222  22222222
```

Figure 12.37 Extended Allen's Operators

In addition to these methods, Allen's operators [100] are also implemented in this system. Conventional Allen's operators are tBefore(), tEqual(), tMeets(), tOverlaps(), tDuring(), tStarts(), and tFinishes() to check the relationships between two intervals (periods) of time as shown in Figure 12.36. Along with these conventional Allen's operators, we enable these operators to perform on two temporal elements each of which is a set of disjoint periods of time as shown in Figure 12.37. In other words, the input of each conventional Allen's operator is an instance of the VTPERIOD temporal data type while an instance of the VTPERIODS temporal data type for each corresponding extended Allen's operator in our system.

Moreover, basic set operations on two periods of time as well as on two sets of periods of time are implemented. These operations are tpUnion(), tpIntersection(), and tpDifference() on two periods of time while tUnion(), tIntersection(), and tDifference() on two sets of disjoint periods of time. The tUnion(), tIntersection(), and tDifference()

operations are closed as their outputs are sets of disjoint periods of time. In contrast, the outputs of the tpUnion(), tpIntersection(), and tpDifference() operations vary from an empty set or NULL to one period to time, or to two periods of time, i.e. a set of disjoint periods of time. Other set operators are defined, e.g. isSubSet(), isEqual(), isMember(), etc. where a period of time is considered as a set of continuous instants (points in time).

Finally, this group includes extra functions GetXXX(SYS.ANYDATA) to extract data of any data type from an instance of the SYS.ANYDATA opaque data type supplied by the employed ORDBMS. Among the methods of this proposed temporal SQL extension module, only these functions are dependent on the support of the employed ORDBMS.

12.3.2.2 Sequenced Methods

The group of sequenced methods includes two parts, the first one curAnyData(), curInsert(), curDelete(), and curUpdate() for temporal upward compatibility and the second one nextAnyData(), prevAnyData(), tSelect(), tInsert(), tDelete(), and tUpdate() for sequenced semantics.

The curAnyData() method is used to extract the value currently true in the modeled world from an attribute history. The curInsert() method is used to insert a value currently true in the modeled world into an attribute history. The curDelete() method is used to remove the value currently true in the modeled world from an attribute history. The curUpdate() method is used to change into a new value the value of an attribute history currently true in the modeled world.

The nextAnyData() method is used to extract from an attribute history the value that is true at the next instant of NOW, i.e. NOW+1, based on some granularity. The prevAnyData() method is used to extract from an attribute history the value that is true at the previous instant of NOW, i.e. NOW-1, based on some granularity. The tSelect() method is used to implement the ASELECT construct defined in chapter 8. The tInsert(), tDelete(), and tUpdate() methods are similarly used to implement temporal insertions, deletions, and updates on an attribute history with some periods of time.

12.3.2.3 Temporal Aggregate Functions

For the support of temporal data aggregation, our system has the temporal aggregate functions defined such as tCOUNT(), tMAX(), tMIN(), tAVG(), and tSUM()

which have already been presented in chapter 8. As of this moment, only the temporal data aggregation of numeric data is supported with the tAVG() and tSUM() functions while the tMAX() and tMIN() functions can be applied to temporal data of a data type with a partial ordering such as alphanumeric and DATE data types and the tCOUNT() function to the aggregation of attribute histories of any data type. These temporal aggregate functions are invoked in the same manner as any built-in or DBMS-supplied non-temporal aggregate functions of the employed ORDBMS.

These temporal aggregate functions are implemented as user-defined aggregate functions supported by the object relational extensibility interface of the employed ORDBMS. The output of each temporal aggregate function is an attribute history defined in subsection 8.2.1. Each temporal composition of the resulting attribute history shows the data aggregation for some periods of time in the modeled world.

12.3.2.4 Translation Stored-Procedures

Translation stored-procedures tsqlCreateType(), tsqlCreateTable(), tsqlAlterType(), tsqlAlterTable(), tsqlDropType(), tsqlDropTable(), tsqlSelect(), tsqlInsert(), tsqlDelete(), and tsqlUpdate() are defined for transmitting temporal SQL statements written in the proposed temporal object relational SQL language in chapter 8 to the employed ORDBMS temporally extended with this temporal data support module. These translation stored-procedures can be directly invoked and executed by the employed ORDBMS via any existing interface in both SQL modes: interactive and embedded.

The tsqlCreateType() procedure takes a CREATE TYPE statement as its input, the tsqlCreateTable() procedure a CREATE TABLE statement, the tsqlAlterType() procedure an ALTER TYPE statement, the tsqlAlterTable() procedure an ALTER TABLE statement, the tsqlDropType() procedure a DROP TYPE statement, the tsqlDropTable() procedure a DROP TABLE statement, the tsqlSelect() procedure a SELECT statement, the tsqlInsert() procedure an INSERT statement, the tsqlDelete() procedure a DELETE statement, and the tsqlUpdate() procedure an UPDATE statement, respectively.

12.3.2.5 External Functions

Corresponding to the tsqlCreateType(), tsqlCreateTable(), tsqlAlterType, tsqlAlterTable, tsqlDropType(), tsqlDropTable(), tsqlSelect(), tsqlInsert(), tsqlDelete(),

and tsqlUpdate() translation stored-procedures, external functions tsqlCreateType_c(), tsqlCreateTable_c(), tsqlAlterType_c(), tsqlAlterTable_c(), tsqlDropType_c(), tsqlDropTable_c(), tsqlSelect_c(), tsqlInsert_c(), tsqlDelete_c(), and tsqlUpdate_c() written in the C programming language are included for transforming a temporal SQL statement into one or many non-temporal SQL statements able to be executed by the employed ORDBMS. Each SQL statement after the translation is temporally enhanced by our temporal data types and methods.

Using the Oracle Call Interface (OCI), a shared library is set up to store the coded C implementation of these external functions. Each of these external functions in turn implements a small token analyzer and parser particularly for its associated statements. During the parsing process, the translation of a temporal SQL statement is carried out.

The result of each function is simply a text that contains the instructions and the transformed SQL statements rewritten with the use of our pre-defined temporal data types and methods. The execution of the transformed SQL statements is then run inside the employed ORDBMS. Hence, the result of a SELECT statement expected to the invoker is not directly returned to the invoker. The invoker needs to further make an appropriate step to have access to the result of the SELECT statement wrapped in the translation stored-procedure that has been invoked.

12.4 Summary

This chapter has presented the temporal object relational database system for temporal 3D objects and their scenes proposed in our research work. The implementation of the proposed system is achieved using the Oracle 10g/11g ORDBMS. However, the employed ORDBMS can be replaced with any other ORDBMS that also supports the extensibility with abstract data types, user-defined methods, user-defined operators, and user-defined optimization techniques. The two main SQL extension modules of the proposed system have been described for temporal data management in general and for temporal 3D objects handling in particular. As of this moment, only the functionalities of the proposed system are focused. Thus, performance issues of the proposed system are not of interest in this dissertation.

Chapter 13

Conclusion

In this dissertation, a temporal object relational database system has been innovatively developed for the handling of 3D objects and their scenes whose properties, relationships, and/or themselves are supported with valid time at both graphical and semantic data levels. The conceptual data representation of the metadata part of temporal 3D objects and their scenes is based on the temporal object oriented NIAM conceptual schema model proposed in chapter 7. The logical data representation of both graphical data and metadata parts of temporal 3D objects and their scenes are based on the object relational data model with the support of valid time proposed in chapter 8. Along with this system, a novel temporal object relational SQL language is invented as a temporal extension of the standard object relational SQL language in association with temporal logic for ease and intuition. The main supplied functionalities to facilitate the handling of temporal 3D objects are summarized as follows.

In chapter 5, a procedure for the handling of 3D objects with valid time is proposed. This procedure shows step by step the achievement of a temporal object relational database for temporal 3D objects from 3D graphical data sources valid for some periods of time in the modeled world.

In chapter 7, the temporal object oriented NIAM conceptual schema model is proposed for the conceptual data representation with temporal aspects and object orientation. Based on this TOONIAM conceptual schema model, a TOONIAM conceptual meta schema is systematically and automatically derived from an input 3D graphical data source with some period of valid time to represent non-temporal/temporal non-explicit/explicit metadata about temporal 3D objects at the abstract level. Integrity constraints are also considered for this derived TOONIAM conceptual meta schema to support the handling of temporal 3D objects.

In chapter 8, the logical temporal data representation is proposed by using the object relational data model with data timetamping at both object and attribute levels. In addition, a temporal object relational SQL language is originally defined with the main functions of a database language including temporal data definitions (schema

definitions, schema evolution definitions, and integrity constraint definitions), temporal data querying, and temporal data modifications. The language is upward compatible with the standard object relational SQL language and temporal upward compatible to enable both non-temporal and temporal database users to work on the same temporal database. It also supports sequenced semantics for temporal data manipulations and allows non-sequenced semantics with non-restrictiveness. The temporal logic is employed in the language for intuitive temporal data manipulations. This language plays a key role in the proposed system to define and manipulate the graphical data and metadata parts of a temporal object relational database for the handling of temporal 3D objects. By means of this language, temporal 3D objects can be accessed in regard to valid time in the SQL standard manner via existing interfaces of the employed ORDBMS at both graphics and semantics levels.

In chapter 9, a temporal object relational database for temporal 3D objects is defined by using the conceptual and logical data representations in the two previous chapters 7 and 8. The graphical data part of temporal 3D objects is fixed for any application domain; thus, handled in the object relational graphical database at the logical data level. In contrast, the metadata part of temporal 3D objects varies from application domain to application domain; thus, considered at both conceptual and logical data levels with a TOONIAM conceptual meta schema and a temporal object relational meta database schema, respectively. The temporal object relational meta database schema can be systematically and automatically obtained from a corresponding TOONIAM conceptual meta schema. Integrity constraints are also introduced to be transformed and enforced along with the temporal object relational meta database schema. All non-temporal/temporal non-explicit/explicit metadata about temporal 3D objects is handled in the temporal object relational meta database. The object relational graphical database and the temporal object relational meta database are brought together to form a temporal object relational database for temporal 3D objects.

In chapter 10, a temporal data population process is determined to automatically populate the data from a 3D graphical data source with valid time into the temporal object relational database defined in chapter 9. The automation of the proposed

temporal data population process can be reached by using the database approach to dealing with temporal 3D objects. Furthermore, the preparation of the proposed temporal data population process is discussed so that this process can be applied in practice with the assumptions of this research.

In conclusion, the achievement of this research is the temporal object relational database system for temporal 3D objects where the querying of temporal 3D objects is enabled in many various aspects: graphics/semantics, non-spatial/spatial, non-presentable/presentable, and non-temporal/temporal by using the proposed temporal object relational SQL language. The original points of the proposed work are stated below as compared to the related works on the handling of 3D objects and their scenes.

- The consideration on the temporal aspect is given not only to 3D objects and their scenes themselves but also to the metadata surrounding them.
- The temporal data representation of both graphical data and metadata parts of 3D objects is invented at both conceptual and logical data levels. Mainly different from the related works, the proposed temporal logical data representation is associated with a new temporal object relational SQL language.
- The temporal database population process on 3D graphical data sources with valid time is considered towards the automation based on the assumed availibility of 3D object regconition. In other works, the population is manual or left as their future work without any temporal aspect of the content of the given sources. Once the temporal aspect of 3D objects is taken into account, the population of these 3D objects into a proper database also needs to be examined with regard to valid time.
- The querying over 3D objects and their scenes is enabled and enriched in a free query approach with many various aspects as mentioned earlier. This can be obtained by means of the proposed temporal object relational SQL language accompanying this system in the same standard manner as the standardized SQL languages. Indeed, many different front-end applications can have access to 3D objects and their scenes for any period of time at any detailed level from object to scene through existing database application programming interfaces. So, sharability and reusability of 3D objects and their scenes can be achieved along the time axis.

Chapter 14

Further Research

A temporal object relational database system has been investigated for the handling of temporal 3D objects and their scenes. The proposed system mainly aims at the functionalities. These functionalities have been detailed in chapters 5, 7, 8, 9, 10, and 11. It is obviously bewared that our system cannot cover every aspect in 3D object management particularly as well as in temporal data management generally. Thus, this chapter will address some further related research of interest.

14.1 On the Proposed System's Side

On the proposed system's side, a further work can be firstly considered for the graphical data support to relax the assumptions of the current work. In particular, *3D object recognition techniques* can be included as part of the proposed system so that the preparation process defined in chapter 10 would not be a burden on those who have to set up the graphics library.

Secondly for the temporal data support, *a built-in semantics support for transaction time* can be achieved by looking into the inherent transaction time support of an existing ORDBMS (Oracle 10g/11g [68] or MS SQL Sever [69]) or by a new transaction time extension. Since transaction time is orthogonal to valid time, the addition of transaction time is feasible and has no impact on the currently achieved system. Apart from transaction time, *a temporal extension of other SQL variants such as views, cursors, and assertions/triggers* is also valuable to facilitate temporal data management at the database level as much as possible instead of letting database users handle their data in respect of time (transaction time as well as valid time) at the application level.

Thirdly, the enrichment of queries supported by the proposed system is more tested *at the graphical data level for a similarity based evaluation*. As of this moment, queries over a scene and its evolutions have been illustrated using the case study in chapter 6. More investigation can be made for *queries over many scenes*.

Fourthly for the efficiency of the proposed system, *performance issues* are promising with *indexing techniques* studied for spatial, temporal, and spatio-temporal

data. These techniques can be trivially implemented and included into our SQL extension modules, which have been defined in chapter 12, through the object relational extensibility interface of the employed ORDBMS.

Finally, we might make an effort to define *the temporal object oriented NIAM conceptual data model* which has been partially introduced in chapter 7 for conceptual temporal data representation. This is because the NIAM conceptual schema model has a strong theoretical foundation with predicate logic and linguistics as compared to the ER and UML models. This topic is associated with the previous one for the querying of a temporal database with temporal logic at the high level by means of a TOONIAM conceptual schema. The temporal logic employed might be the temporal predicate logic that needs to be formally defined. Also, attention should be given to the support for *temporal integrity constraints*, especially temporal explicit integrity constraints, along with a TOONIAM conceptual schema. As of this moment, the specification of temporal integrity constraints at the conceptual data level is limited. Moreover, the transformation of these constraints into the ones at the logical data level needs more consideration as such temporal explicit integrity constraints are normally left on the application's side.

14.2 On the Application's Side

At this moment, our system is constructed along with the case study in chapter 6 for functionality illustration purposes. As realized previously, the case study is small and simple enough to be used for an illustration of conceptual and logical data representations, of data population, and of data manipulations on the graphical data and metadata parts of 3D objects and their scenes over the time. Since this research does not concentrate on performance issues, the case study has no impact on the currently achieved system. However, practical applications are certainly of interest to be investigated with the system.

Therefore, some further research on the application's side should be taken into account. This subsection points out several potential applications that can be developed on top of the proposed system. Others can be added up to whether the users can take advantage of the system for their own benefits. *Here are only a few potential applications listed.*

(i) The handling of 2D/3D medical images

In [113], spatial and temporal aspects are realized for feature and content-based medical image retrieval. Medical images handled by their system are of two dimensions. Nowadays, 3D medical images are of growing interest with advances in medical technology [114]. Thus, our proposed system is potentially utilized for the handling of 3D medical images with temporal/3D spatial data supports.

(ii) Video content management

Developed as a general-purpose temporal object relational database system, the proposed system is able to be used for video content management like the one introduced in [115]. Each 3D scene true for some period of time in the modeled world is now replaced by a video keyframe at some moment. Objects of interest in that keyframe are recognized and then typified in order to be populated into a temporal object relational database managed by our system. With the support for temporal/3D data along with 3D spatial functions, it is believed that spatio-temporal queries illustrated in [115] can be answered by our system. In other words, the proposed system can take care of video contents.

(iii) Robotic assistants

In the future, robotic assistants will get popular. Different from human-beings, robotic assistants need to be instructed at least at the beginning. In order to understand the surrounding environment, robotic assistants must be enabled to analyze their captured real 3D scene. If expected to answer any question, they must be enabled to make queries over the information in hand. In [116], the first ability is achieved with a 3D object recognition algorithm on 3D cluttered scenes for autonomous household robots. That would be nice if such a robot can answer the questions related to spatial, temporal, and/or spatio-temporal aspects such as what have been here for five days, when the last guest came to the house, who stood near the window yesterday, etc. Those robots can take advantage of the capability of temporal/3D spatial data querying in our system.

(iv) Virtual environment exploration

Among the related works on the handling of 3D objects in a virtual environment, we pay attention to some of their applications for virtual environment exploration. [9] intends to support navigation by query. [10] allows virtual exhibition. [11] aims at web search engines that process requests formulated in the natural language referring to high-level features; extraction of semantic objects from huge files for easier examination; and automatic creation of high-level libraries used to create different 3D environments, high-level textual descriptions of paths to a given location that are generated on the basis of static paths or dynamically generated paths. As already bewared in [117], the proposed system can play a role of the underlying platform for so-called information-rich virtual environments.

Bibliography

[1]. S-F. Chang, T. Sikora, and A. Puri, Overview of MPEG-7 Standard, IEEE Trans. Circuits System Video Technology 11(2001) 688-695.

[2]. Geographic Information – Metadata, International Standard ISO 19115:2003, 2003.

[3]. Dublin Core, http://dublincore.org/.

[4]. C. Toro, J. Posada, S. Wundrak, A. Stork, Improving Virtual Reality Applications in CAD through Semantics, The International Journal of Virtual Reality 5 (2006) 39-46.

[5]. I. M. Bilasco, M. Villanova-Oliver, J. Gensel, H. Martin, 3DSEAM: a Model for Annotating 3D Scenes Using MPEG-7, in: Proc. the 7th IEEE Intl. Symposium on Multimedia, IEEE, 2005, pp. 310-319.

[6]. I. M. Bilasco, J. Gensel, M. Villanova-Oliver, H. Martin, An MPEG-7 framework enhancing the reuse of 3D models, in: Proc. the 11th International Conference on 3D Web Technology, 2006, pp. 65-74.

[7]. P. Halabala, Semantic Metadata Creation, in: Proc. CESCG: 7th Central European Seminar on Computer Graphics, 2003, pp. 15-25.

[8]. E. Kalogerakis, S. Christodoulakis, N. Moumoutzis, Coupling Ontologies with Graphics Content for Knowledge Driven Visualization, in: Proc. the IEEE Virtual Reality Conf. (VR 2006), IEEE, 2006, pp. 43-50.

[9]. J. I. Martinez, C. D. Mata, A Basic Semantic Common Level for Virtual Environments, The Intl. Journal of Virtual Reality 5 (2006) 25-32.

[10]. K. A. Otto, The Semantics of Multi-user Virtual Environments, in: Proc. the Workshop towards Semantic Virtual Environments (SVE'05), 2005.

[11]. F. Pittarello, A. De Faveri, Semantic Description of 3D Environments: a Proposal Based on Web Standards, in: Proc. the 11th Intl. Conf. on 3D Web Technology (Web3D 2006), 2006, pp.85-95.

[12]. J. H. Hwang, J. H. Ro, K. H. Lee, J. S. Park, S. Hwang, An XML-based 3-dimensional graphic database system, in: Proc. The First Intl. Conf. the Human Society and the Internet, 2001, pp. 454-467.

[13]. Y. Kim, S. Hwang, A content-based 3D graphic information retrieval system, in: Proc. The Third Asia Information Retrieval Symp., 2006, pp. 567-573.

[14]. E. F. Codd, A relational model for large shared data banks, Communications of the ACM 13 (1970) 377-387.

[15]. M. A. Roth, H. F. Korth, A. Silberschatz, Extended Algebra and Calculus for Nested Relational Databases, ACM Trans. Database Systems 13 (1988) 389-417.

[16]. D. Bartels, ODMG 93-the emerging object database standard, in: Proc. the 12th IEEE International Conference on Data Engineering, IEEE, 1996, pp. 674-676.

[17]. ANSI/ISO/IEC International Standard (IS) 9075 – Database Language SQL:2003 - WD 9075-2 (SQL/Foundation), Sep 2003.

[18]. M. Stonebraker, P. Brown with D. Moore, Object-Relational DBMS's – Tracking the Next Great Wave (Morgan Kaufmann, 1999).

[19]. C. S. Jensen et al., The consensus glossary of temporal database concepts, in Temporal Databases: Research and Practice, eds. O. Etzion, S. Jajodia, S. Sripada, (Springer-Verlag, Berlin, 1998), pp. 367-405.

[20]. D. Dey, T. M. Barron, and V. C. Storey, A complete temporal relational algebra, The VLDB Journal 5 (1996) 167-180.

[21]. E. A. Emerson, Temporal and Modal Logic, Handbook of Theoretical Computer Science, J. van Leeuwen, ed., vol. 2, 1990.

[22]. Virtual Reality Modeling Language, http://www.web3d.org/x3d/specifications/#vrml97.

[23]. X3D, http://www.web3d.org/x3d/.

[24]. T. Griffiths, A. A. A. Fernandes, N. W. Paton, R. Barr, The Tripod spatio-historical data model, Data & Knowledge Engineering 49 (2004) 23-65.

[25]. B. Huang and C. Claramunt. STOQL: An ODMG-based Spatio-Temporal Object Model and Query Language, in: Proc. the 10th Int'l Symposium on Spatial Data Handling, 2002.

[26]. S. Shumilov, J. Siebeck, Database support for temporal 3D data: Extending the GeoToolKit, in: Proc. the 7th EC-GI & GIS Workshop, 2001.

[27]. A. Vakaloudis, B. Theodoulidis, The Dynamic Construction of VRML Worlds by the Use of a VRML Database Repository, IEE Seminar Digests 1999 4 (1999) 4/1-4/7.

[28]. J. R. Viqueira, N. A. Lorentzos, SQL Extension for Spatio-Temporal Data, The Very Large Database Journal 16 (2007) 179-200.

[29]. M. F. Worboys, A Unified Model for Spatial and Temporal Information, The Computer Journal 37 (1994) 26-34.

[30]. C. Arens, J. Stoter, P. v. Oosterom, Modelling 3D spatial objects in a geo-DBMS using a 3D primitive, Computers & Geosciences 31 (2005) 165-177.

[31]. G. Gröger, M. Reuter, L. Plümer, Representation of a 3-D City Model in Spatial Object-Relational Databases, in: International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 34 (2004). Part B4 (Proc. of the XXth ISPRS Congress, Turkey).

[32]. M. Schneider, B. E. Weinrich, An Abstract Model of Three-Dimensional Spatial Data Types, in: Proc. 12th ACM Int. Symp. on Advances in Geographic Information Systems, ACM, 2004, pp. 67-72.

[33]. S. Zlatanova, 3D geometries in spatial DBMS, in: Proc. International Workshop on 3D GeoInformation (GeoInfo '06), 2006.

[34]. E. Paquet, M. Rioux, A Content-Based Search Engine for VRML Databases, in: Proc. the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, 1998, pp. 541-546.

[35]. A. S. Mian, M. Bennamoun, R. Owens, Three-dimensional model-based object recognition and segmentation in cluttered scenes, IEEE Trans. Pattern Analysis and Machine Intelligence 28 (2006) 1584-1601.

[36]. A. D. Bimbo, P. Pala, Content-based retrieval of 3D models, ACM Trans. Multimedia Computing, Communications and Applications 2 (2006) 20-43.

[37]. E. Onasoglou, D. Katsikas, A. Mademlis, P. Daras, M. G. Strintzis, A novel prototype for documentation and retrieval of 3D objects, in: Proc. 14th Intl. Conf. on Image Analysis and Processing Workshops, ICIAPW2007, IEEE, 2007, pp. 89-94.

[38]. L. Dietze, U. Nonn, A. Zipf, Metadata for 3D City Models: Analysis of the Applicability of the ISO 19115 Standard Possibilities for further Amendments, in: Proc. the 10th AGILE International Conference on Geographic Information Science, 2007.

[39]. M.E. Latoschik, M. Schilling, Incorporating VR Databases into AI Knowledge Representations: A Framework for Intelligent Graphics Applications, in: Proc. the 6th IASTED Intl. Conf. on Computer Graphics and Imaging, 2003, pp. 79-84.

[40]. Geographic Information – Spatial Schema, International Standard ISO 19107:2003, 2003.

[41]. Resource Description Framework (RDF), http://www.w3.org/RDF/.

[42]. A. U. Tansel, Temporal relational data model, IEEE Transactions on Knowledge and Data Engineering 9 (3) (1997) 464-479.

[43]. S. K. Gadia, A homogeneous relational model and query languages for temporal databases, ACM Transactions on Database Systems 13 (4) (1988) 418-448.

[44]. M. H. Böhlen, Valid time integrity constraints, Technical Report, TimeCenter TR 94-30, Department of Computer Science, University of Arizona, 1994.

[45]. M. H. Böhlen, C. S. Jensen, R. T. Snodgrass, Temporal statement modifiers, ACM Transactions on Database Systems 25 (4) (2000) 407-456.

[46]. M. Böhlen, R. Snodgrass, M. Soo, Coalescing in temporal databases, Technical Report, TimeCenter TR-9, 1997; also in: Proc. the 22nd VLDB Conference, 1996, pp. 180-191.

[47]. A. Carvalho, C. Ribeiro, A. A. Sousa, A spatio-temporal database system based on TimeDB and Oracle Spatial, in: Proc. IFIP International Federation for Information Processing, 2006, pp. 11-20.

[48]. C. X. Chen, J. Kong, and C. Zaniolo, Design and implementation of a temporal extension of SQL, in: Proc. ICDE, IEEE, 2003, pp. 689-691.

[49]. N. A. Lorentzos, Y. G. Mitsopoulos, SQL extension for interval data, IEEE Transactions on Knowledge and Data Engineering 9 (3) (1997) 480-499.

[50]. R. T. Snodgrass, The temporal query language TQuel, ACM Trans. on Database Systems 12 (2) (1987) 247-298.

[51]. R. T. Snodgrass, I. Ahn, G. Ariav, D. Batory, J. Clifford, C.E. Dyreson, R. Elmasri, F. Grandi, C.S. Jensen, W. Kä, TSQL2 language specification, SIGMOD Record 23 (1) (1994) 65-86.

[52]. R. T. Snodgrass, Managing temporal data: a five-part series, Technical Report TR-28, Department of Computer Science, University of Arizona, 1998.

[53]. J. Yang, H. Ying, J. Widom, TIP: a temporal extension to Informix, in: Proc. SIGMOD, 2000, pp. 596-671.

[54]. C. Combi, A. Montanari, G. Pozzi, The T4SQL Temporal Query Language, in: Proc. CIKM'07, ACM, 2007, pp. 193 – 202.

[55]. M. H. Böhlen, Temporal database system implementations, SIGMOD Record 24 (1995) 53-60.

[56]. A. U. Tansel, C. E. Atay, Nested bitemporal relational algebra, in: Proc. ISCIS 2006, LNSC 4263, Springer, 2006, pp. 622-633.

[57]. M. Dumas, M. C. Fauvet, and P. C. Scholl, TEMPOS: a platform for developing temporal applications on top of object DBMS, IEEE Transactions on Knowledge and Data Engineering 16 (3) (2004) 354-374.

[58]. N. Edelweiss, P. N. Hübler, M. M. Moro, G. Demartini, A Temporal Database Management System Implemented on Top of a Conventional Database, in: Proc. The XX International Conference of the Chilean Computer Science Society (SCCC'00), IEEE, 2000.

[59]. R. Mata-Toledo, M. Monger, Implementing a temporal data management system within Oracle, Journal of Computing Sciences in Colleges 23 (2008) 76-81.

[60]. A. Sotiropoulou, M. Souillard, C. Vassilakis, Temporal extension to ODMG, in: Proc. IADT'98 – Issues & Applications of Database Technology, 1998, pp. 304-311.

[61]. M. A. Bassiouni, A logic for handling time in temporal databases, in: Proc. COMPSAC, IEEE, 1988, pp. 345-352.

[62]. L. Fegaras and R. Elmasri, A temporal object query language, in: Proc. TIME, IEEE, 1998, pp. 51-57.

[63]. G. Garani, A generalized temporal algebra, Data & Knowledge Engineering 57 (2006) 283-310.

[64]. C. Kleiner et al., Natural and efficient modeling of temporal information with object-relational databases, Technical Report, University of Hannover, 2002.

[65]. C. Kleiner, U. W. Lipeck, Performance of querying temporal attributes in object-relational databases, in: Proc. TIME'02, IEEE, 2002, pp. 58-60.

[66]. I. Merlo, E. Bertino, E. Ferrari, S. Gadia, G. Guerrini, Querying multiple temporal granularity data, in: Proc. TIME'00, IEEE, 2000, pp. 103-114.

[67]. ISO/IEC JTC 1/SC 32 N 0436, Rationale for the Withdrawal of Projects, 2000.

[68]. Oracle Flashback Query, Application developer's guide – fundamentals, 10g release 1(10.1), No. B10795-01.

[69]. D. Lomet, R. Barga, M. Mokbel, G. Shegalov, R. Wang, Y. Zhu, Transaction time support inside a database engine, in: Proc. ICDE Conference, IEEE, 2006, pp. 35-46.

[70]. Oracle Valid Time Support, Application developer's guide – workspace manager, 10g release 1(10.1), No. B10824-01.

[71]. Oracle 10g/11g, http://www.oracle.com/.

[72]. K. Nørvåg, M. Limstrand, L. Myklebust, TeXOR: temporal XML database on an object relational database system, in: Proc. Perspectives of System Informatics, Springer, 2003, pp. 520-530.

[73]. F. Wang, C. Zaniolo, An XML-based approach to publishing and querying the history of databases, World Wide Web: Internet and Web Information Systems 8 (3) (2005) 233-259.

[74]. R. Elmasri, S. B. Navathe. Fundamentals of Database Systems. Addison Wesley, 2003.

[75]. P. Chen, The entity-relationship model – toward a unified view of data, ACM Transactions on Database Systems (TODS) 1 (1976) 9-36.

[76]. Unified Modeling Language (UML), http://www.uml.org/.

[77]. G. M. Nijssen, T. A. Halpin, Conceptual Schema and Relational Database Design: A Fact-oriented Approach, Prentice Hall, 1989.

[78]. E. F. Codd, Data Models in Database Management, in: Proc. Data Abstraction, Databases workshop, 1980.

[79]. K. Niyomthum, S. Chittayasothorn, A transformation from an object database to an object relational database, in: Proc. SoutheastCon, IEEE, 2003.

[80]. T. B. Gendreau, Object Oriented Extensions to SQL, MICS, 2005.

[81]. A. Eisenberg, J. Melton, K. Kulkarni, J. E. Michels, F. Zemke, SQL:2003 Has Been Published, ACM Sigmod Record, 2004.

[82]. A. Eisenberg, J. Melton, SQL Standardization: The Next Steps, ACM Sigmod Record, 2000.

[83]. A. Steiner and M. C. Norrie. Implementing Temporal Database in Object-Oriented Systems, in: Proc. DASFAA'97, 1997.

[84]. J. Green, R. Johnson, ProSQL: A Prototyping Tool for SQL Temporal Language Extensions, in: Proc. The 20th British National Conference on Databases, Springer-Verlag, 2003.

[85]. V. S. Lai, J. Kuilboer, J. L. Guynes, Temporal Databases: Model Design and Commercialization Prospects, DATABASE 25 (3) (1994).

[86]. R. T. Snodgrass, M. Bohlen, C. Jensen, and A. Steiner. Transitioning Temporal Support in TSQL2 to SQL3, in: Temporal Databases: Research and Practice, 1998.

[87]. T. Sellis. CHOROCHRONOS – Research on Spatio-Temporal Database Systems, in: Proc. ISD'99, LNCS1737, Springer-Verlag, 1999.

[88]. Autodesk 3ds Max, AutoCAD, http://usa.autodesk.com/.

[89]. 3D Modeling Software – trueSpace, http://www.caligari.com/.

[90]. Internet Space Builder, http://www.parallelgraphics.com/products/isb/.

[91]. Direct3D, http://msdn.microsoft.com/msdnmag/issues/03/07/DirectX90/default.aspx.

[92]. OpenGL, http://www.opengl.org/.

[93]. Java3D, http://java.sun.com/products/java-media/3D/.

[94]. The Web3D Consortium, http://www.web3d.org.

[95]. GeoVRML, http://www.ai.sri.com/geovrml/1.1/doc/.

[96]. H. Luttermann and M. Grauer, VRML History: Storing and Browsing Temporal 3D-Worlds, in: Proc. the 4th Symposium on VRML, ACM-Press, 1999.

[97]. VRML-History, http://www-winfo.uni-siegen.de/vrmlHistory/docs/index.html.

[98]. Open Geospatial Consortium, Inc., OpenGIS® Geography Markup Language (GML) – Implementation Specification, Version 3.1.1, 2004.

[99]. CityGML, http://www.citygml.org/.

[100]. J. F. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM 26 (1983) 832-843.

[101]. C. S. Jensen, J. Clifford, S. K. Gadia, F. Grandi, et al., The TSQL benchmark, in: Proc. the International Workshop on an Infrastructure for Temporal Databases, 1993, pp. QQ-1-QQ-28.

[102]. N. Pissinou, R. T. Snodgrass, R. Elmasri, et al., Towards an infrastructure for temporal databases: report of an invitational ARPA/NSF workshop, Technical Report, University of Arizona, 1994.

[103]. R. M. Galante, C. S. dos Santos, N. Edelweiss, Á. F. Moreira, Temporal and versioning model for schema evolution in object-oriented databases, Data & Knowledge Engineering 53(2005) 99-128.

[104]. M. M. Moro, N. Edelweiss, A. P. Zaupa, C. S. dos Santos, TVQL – temporal versioned query language, in: Proc: DEXA 2002, Springer-Verlag, 2002, pp. 618-627.

[105]. A. U. Tansel, Temporal data modeling and integrity constraints in relational databases, in: Proc. Symposium on Computer and Information Sciences, LNCS 3280, Springer, 2004, pp. 459-469.

[106]. M. H. Böhlen, J. Gamper, C. S. Jensen, How would you like to aggregate your temporal data?, in: Proc. TIME'06, IEEE, 2006, pp. 121-136.

[107]. E. Zimányi, Temporal aggregates and temporal universal quantification in standard SQL, SIGMOD Record 35 (2) (2006) 16-21.

[108]. K. Torp, C. S. Jensen, M. Böhlen, Layered temporal DBMS's – concepts and techniques, in: Proc. DASFAA'97, 1997, pp. 371-380.

[109]. C. Vassilakis, P. Georgiadis, A. Sotiropoulou, A comparative study of temporal DBMS architectures, in: Proc. DEXA'96, IEEE, 1996, pp. 153-164.

[110]. E. Malinowski, E. Zimányi, A conceptual model for temporal data warehouses and its transformation to the ER and the object-relational models, Data & Knowledge Engineering 64 (2008) 101-133.

[111]. Oracle 10g (10.1), Oracle® Spatial: User's Guide and Reference, B10826-01, Dec 2003.

[112]. Open GeoSpatial Consortium, OpenGIS Simple Features Specification for SQL (SFS), 1999.

[113]. W. W. Chu, C-C Hsu, A. F. Cárdenas, R. K. Taira, Knowledge-based image retrieval with spatial and temporal constructs, IEEE Transactions on Knowledge and Data Engineering 10 (6) (1998) 872-888.

[114]. T. S. Yoo, 3D medical informatics: information science in multiple dimensions, Chapter 12 in: Medical Informatics: Knowledge Management and Data Mining in Biomedicine, eds. H. Chen, S. S. Fuller, C. Friedman, and W. Hersh, Springer, 2005, pp. 333-358.

[115]. M. E. Dönderler, E. Saykol, Ö. Ulusoy, U. Güdükbay, BilVideo: a video database management system, IEEE Multimedia (2003) 66-70.

[116]. R. B. Rusu, N. Blodow, Z. Marton, A. Soos, M. Best, Towards 3D object maps for autonomous household robots, in: Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, IROS 2007, IEEE, 2007, pp. 3191-3198.

[117]. D. A. Bowman, C. North, J. Chen, N. F. Polys, P. S. Pyla, U. Yilmaz, Information-rich virtual environments: theory, tools, and research agenda, in: Proc. the ACM Symposium on Virtual Reality Software and Technology, VRST03, ACM, 2003, pp. 81-90.

Appendix A

User's Manual of the Extension to an Employed ORDBMS for Temporal and 3D Data Supports

In this section, the user's manual of the extension to an employed ORDBMS, Oracle 10g/11g in this work, for temporal and 3D data supports is briefly presented.

A.1 System Requirements

- The operating system (Microsoft Windows): Windows 2000 Advanced Server or Windows XP Professional or Windows 2003
- The Oracle ORDBMS: 10g or 11g

A.2 The Deployment of the Extension

The extension is developed as an Oracle 10g/11g data cartridge that can be directly and easily integrated as part of the employed ORDBMS. The cartridge name is "TSQL Data Cartridge", the cartridge ID is "T3D", the schema/owner under which the extension is defined is "T3D".

The procedure for deploying the extension for temporal and 3D data supports is step by step executed as follows.

A.2.1 Create the Cartridge Schema

- Connect as SYSTEM AS SYSDBA;
- Create user T3D identified by T3D;
- Grant appropriate privileges to T3D;
- Modify the Network Configuration file: ./network/admin/listener.ora in order that external routines in the proposed dynamic link libraries (DLL) can be invoked.

A.2.2 For Temporal Data Support

- Connect as SYS as SYSDBA for the installation of the temporal data cartridge;
- Execute dbms_registry.loading in order to load all components of the cartridge;
- Define types;

- Define meta tables;
- Define the functions GetXXX() to extract data of some data type from an instance of the SYS.ANYDATA opaque type;
- Define the main package 'tsql' that contains all functions related to temporal upward compatibility and temporal data manipulation supports;
- Define the temporal aggregate functions;
- Define the dynamic link libraries and external methods for statement translations;
- Define the translation store-procedures;
- Grant privileges on these components so that other users can benefit them;
- Create the validate procedure in order to check the validity of the cartridge after the deployment;
- Execute dbms_registry.loaded to finish loading the components of the cartridge;
- Execute dbms_registry.valid to set the status of the cartridge to 'valid' so that it can be used by other users;
- Execute the validate procedure for a final validity check of the cartridge deployment.

A.2.3 For 3D Data Support

- Connect as T3D;
- Create user-defined types corresponding to VRML field and node types for graphical databases;
- Create meta and graphical tables;
- Create the functions GetXXX() to extract data of a graphical data type from an instance of the SYS.ANYDATA opaque type;
- Create the equality check functions on the user-defined types corresponding to the VRML node types;
- Create the type bodies of the user-defined types corresponding to the VRML node types each of which contains the constructor functions and member functions (toVRML() and compBBox());
- Create 3D spatial functions;
- Grant proper privileges on these types and functions to other users.

A.3 The Usage of the Extension

This proposed SQL extension for temporal and 3D data supports can be utilized exactly in the same way as any DBMS-supplied cartridge. All the data types and methods in the extension can be used in two SQL modes: interactive and embedded.

Working in the interactive mode can be done by using the Oracle SQL *Plus end-user interface supplied by the Oracle ORDBMS. It is also enabled with the prototype of the proposed temporal object relational database system for temporal 3D objects. Using the dialog "Temporal Data Manipulation over Temporal 3D Objects/Scenes" of this prototype, a non-temporal/temporal SQL statement specified in chapter 8 can be directly submitted to the proposed system without being wrapped in any translation stored-procedure that has been previously introduced in chapter 12.

Working in the embedded mode can be done with any host language (programming language) that is supported by the employed ORDBMS, the Oracle 10g/11g ORDBMS in this work. A non-temporal/temporal SQL statement on a temporal/3D database must be wrapped in an appropriate translation stored-procedure as previously mentioned while a non-temporal SQL statement on a non-temporal/3D database is processed as usual. The invocation of a translation stored-procedure and its execution is exactly the same as the one of any stored-procedure defined on the employed ORDBMS. The feasibility of the employment of the proposed extension has been proved with the successful development of the prototype of the proposed temporal object relational database system for temporal 3D objects where the temporal object relational SQL language proposed in chapter 8 is utilized to have access to temporal/3D databases. This prototype will be detailed in the next appendix.

Appendix B

A Prototype of the Proposed Temporal Object Relational Database System for 3D Objects

A prototype of the proposed temporal object relational database system for 3D objects is introduced in this appendix. The prototype is built by using the Visual Basic .Net programming language on the temporal ORDBMS, which is achieved from the extension of the employed Oracle 10g/11g ORDBMS with the proposed temporal 3D data cartridge presented in the previous appendix. The temporal object relational SQL language defined in chapter 8 is used during the development of this prototype within the .NET framework.

B.1 System Requirements

- The operating system: Windows 2000 Advanced Server or Windows XP Professional or Windows 2003
- The Oracle ORDBMS: 10g or 11g
- The Microsoft .NET framework is also required.

B.2 Functionalities

The prototype is developed with the following main components.

- ***The 3D graphical library file loader***: any VRML library file that contains a 3D graphical pattern associated with some semantic object type is input and processed. The graphical data population process is invoked for the processing of VRML library files.
- ***The 3D graphical data source file importer***: any VRML data source file that contains a collection of 3D objects forming a 3D scene is input with a period of time for the validity of the graphical data content in the modeled world. The input file will be processed using the previously supplied 3D graphical library files so that 3D objects in a scene are extracted according to their appropriate semantic object type. The minimum bounding box of each extracted object is also computed. Graphical descriptions of 3D objects are kept in the graphical part and extracted explicit metadata about 3D objects in the

metadata part of the temporal object relational database with regard to valid time. The objectifying process, graphical data population process, and temporal metadata population processes are invoked while the input graphical data source file is being processed.

- *The 3D graphical target file exporter*: 3D objects of a scene are able to be retrieved for some period of time during which those objects are valid in the modeled world. Their VRML descriptions can be reconstructed and exported into a VRML file. This target VRML file can be processed by any VRML browser to display the retrieved 3D objects.
- *The interactive temporal data manipulation processor*: This component allows an SQL statement to be issued and executed in the interactive mode without being wrapped in any proposed translation stored-procedure. This support also enables non-temporal/temporal data manipulations on the non-explicit metadata part of a temporal object relational database for temporal 3D objects and their scenes along time.

B.3 User Interfaces

In this subsection, a few screenshots are provided to introduce the user interface of the prototype and to illustrate the use of the proposed temporal object relational database system for the handling of temporal 3D objects and their scenes.

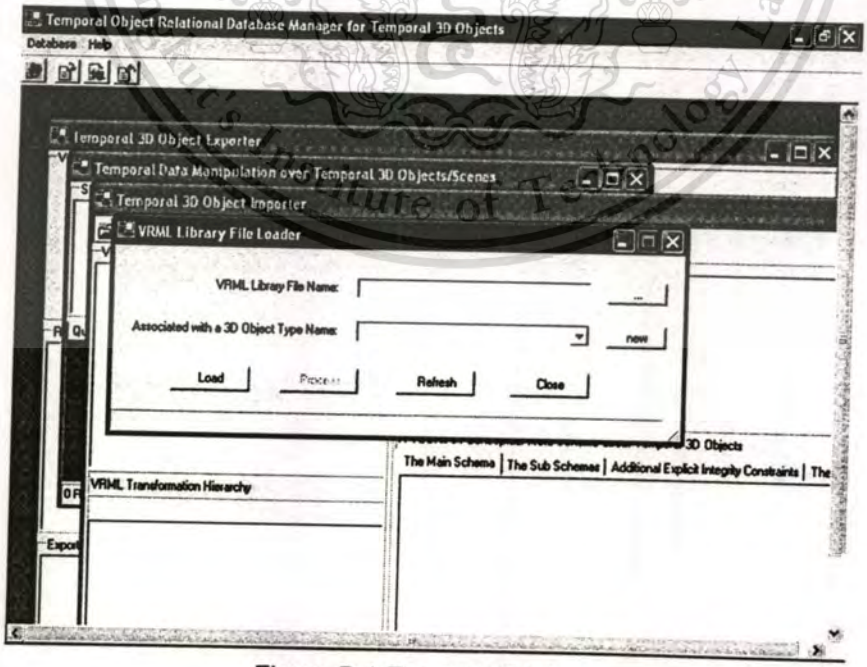


Figure B.1 The Main Dialogs

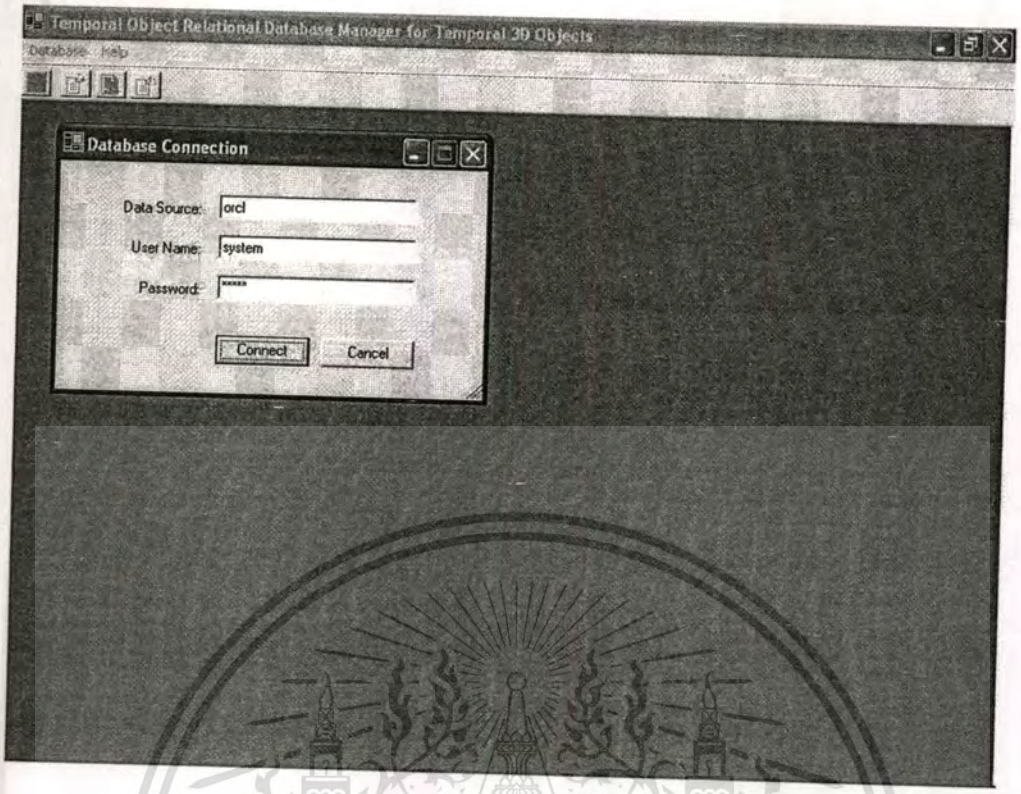


Figure B.2 Database Connection

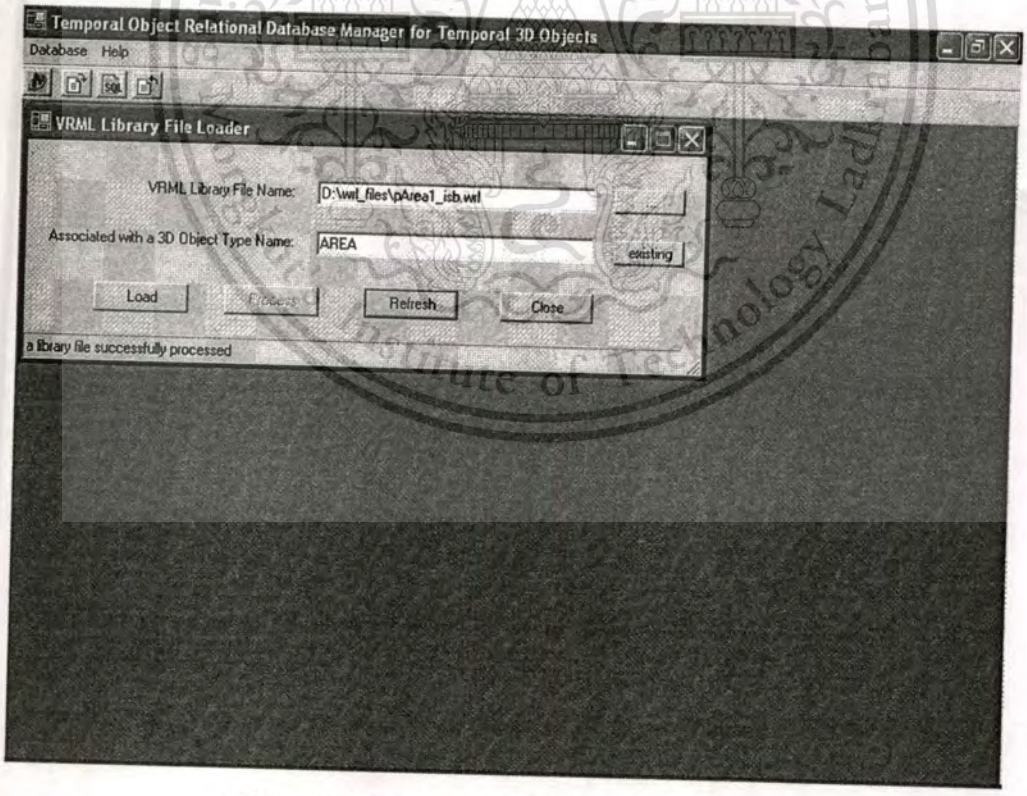


Figure B.3 The Graphical Library File Loader

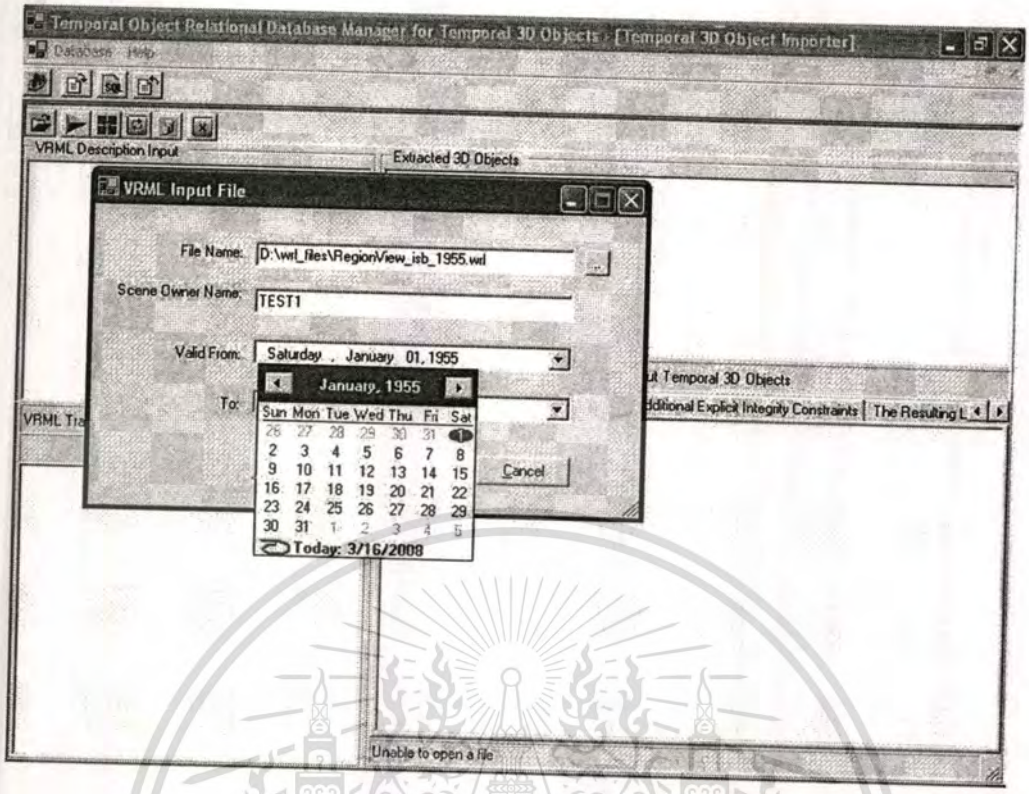


Figure B.4 The Input for the Graphical Source File Importer

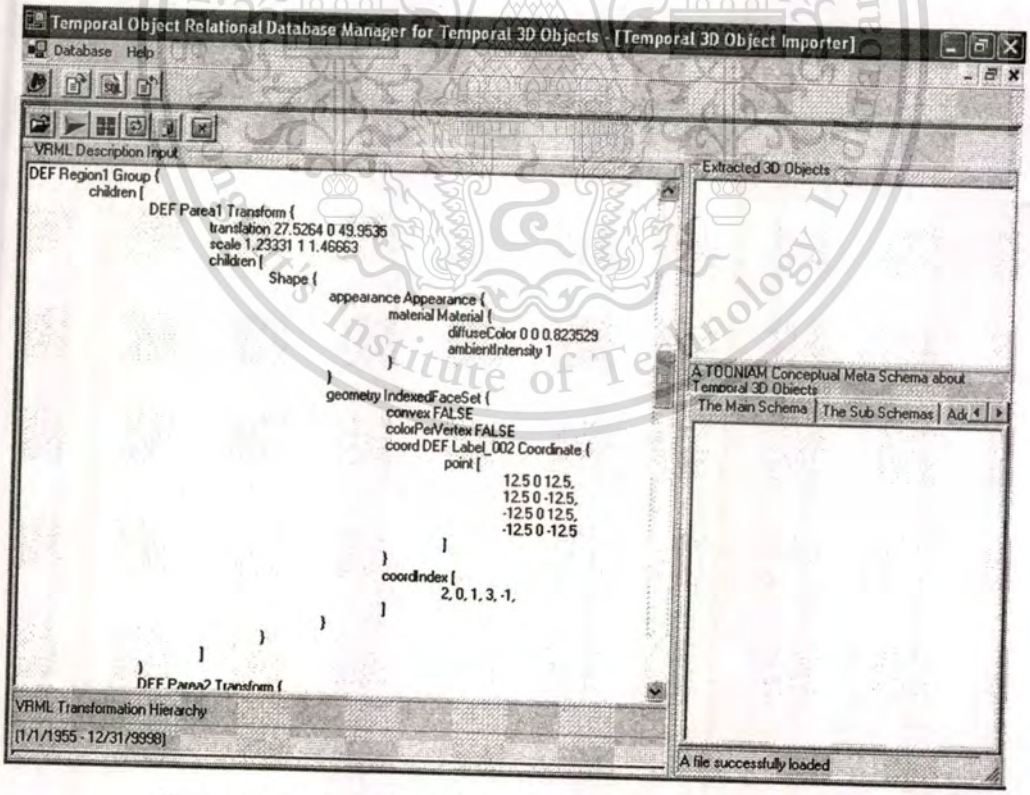


Figure B.5 The Content of the Input Graphical Source File

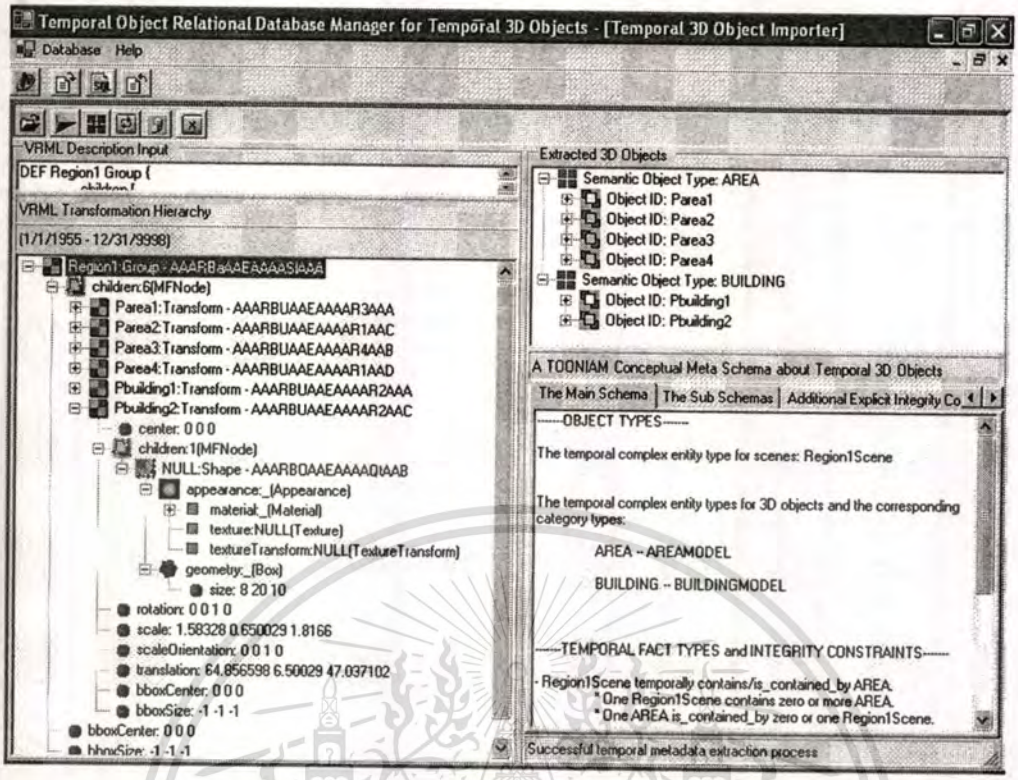


Figure B.6 The Output of the Objectifying Process and the Establishment of the TOONIAM Conceptual Meta Schema

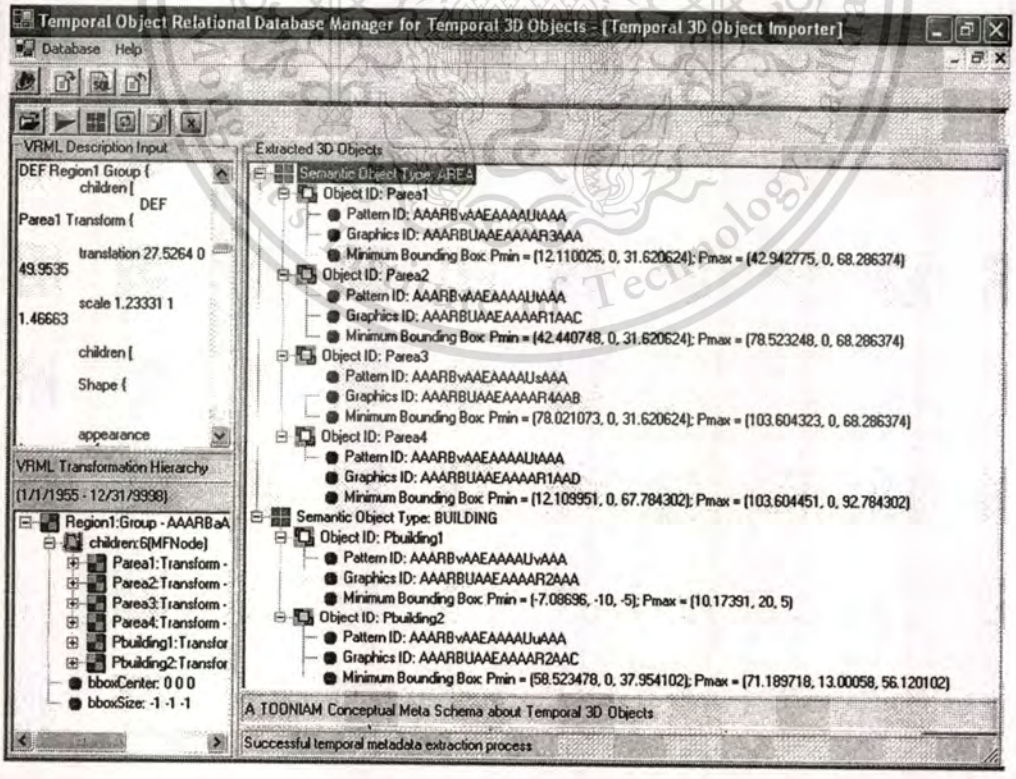


Figure B.7 The Extracted Objects

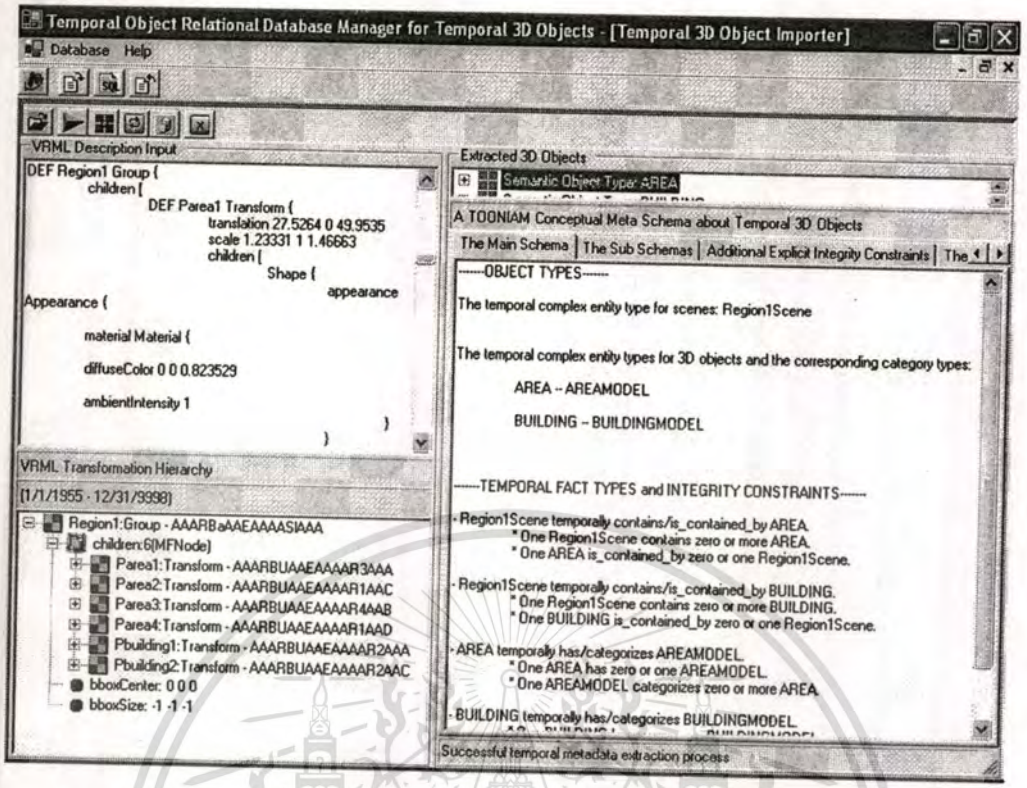


Figure B.8 The Specification of the Main Schema of the TOONIAM Conceptual Meta Schema

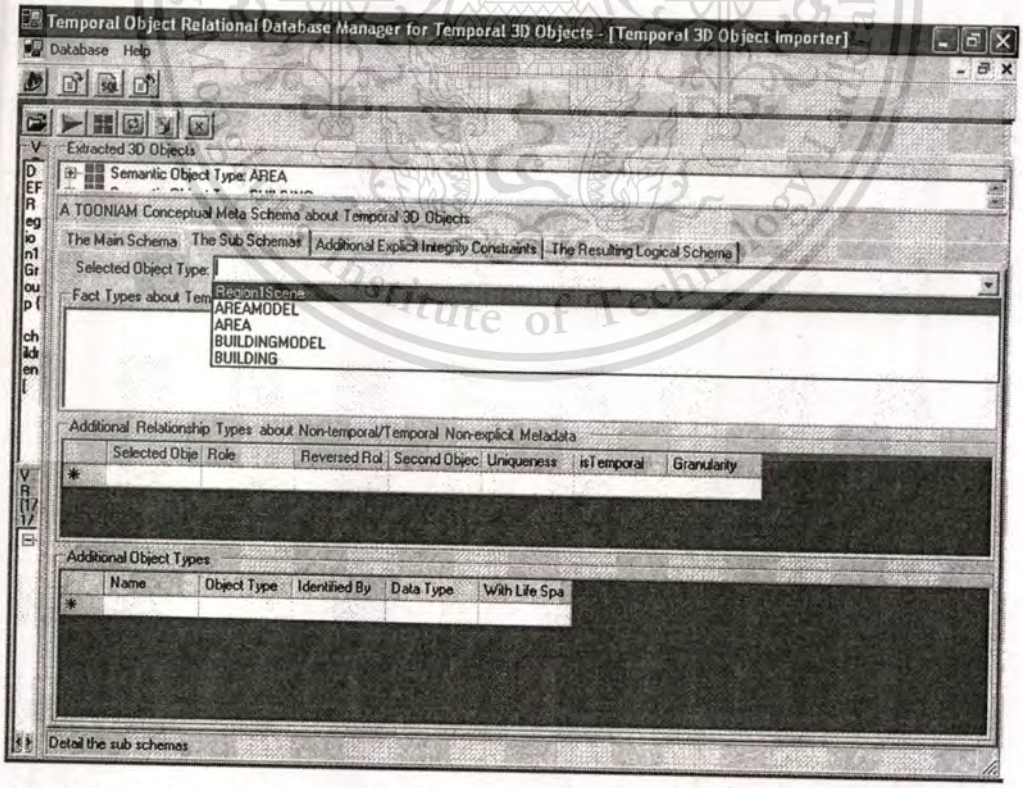


Figure B.9 The Extracted Semantic Object Types of the TOONIAM Conceptual Meta Schema

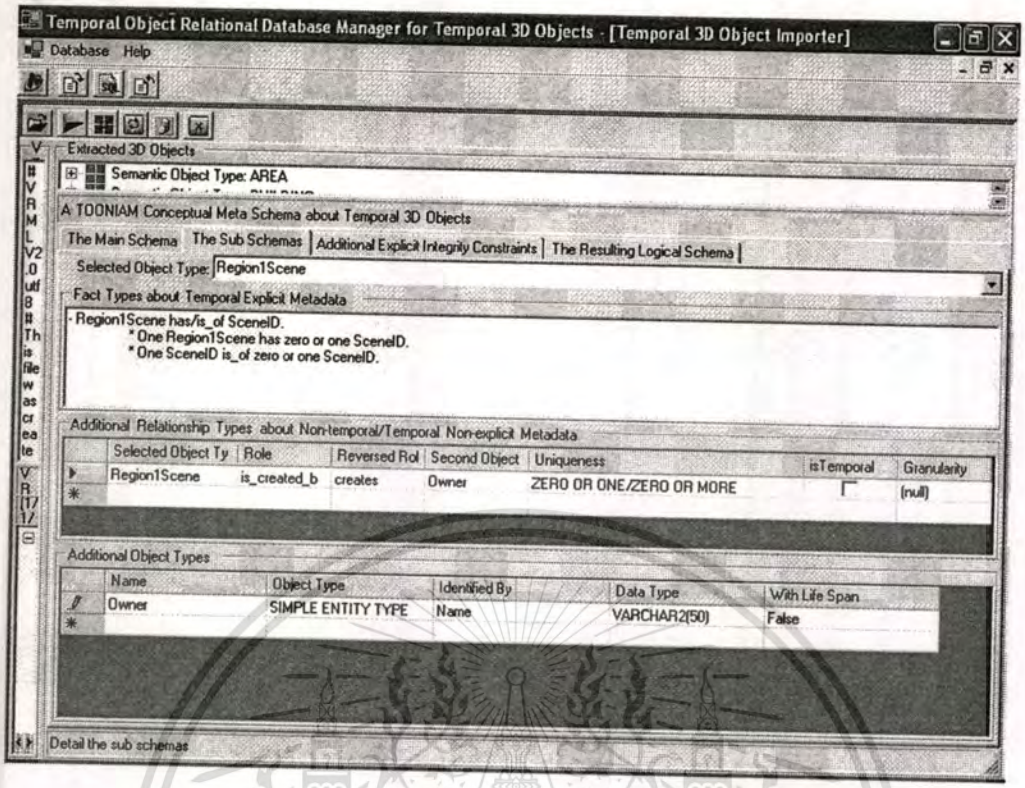


Figure B.10 The Detailing of the Sub Schema of the Temporal Complex Entity Type Region1Scene

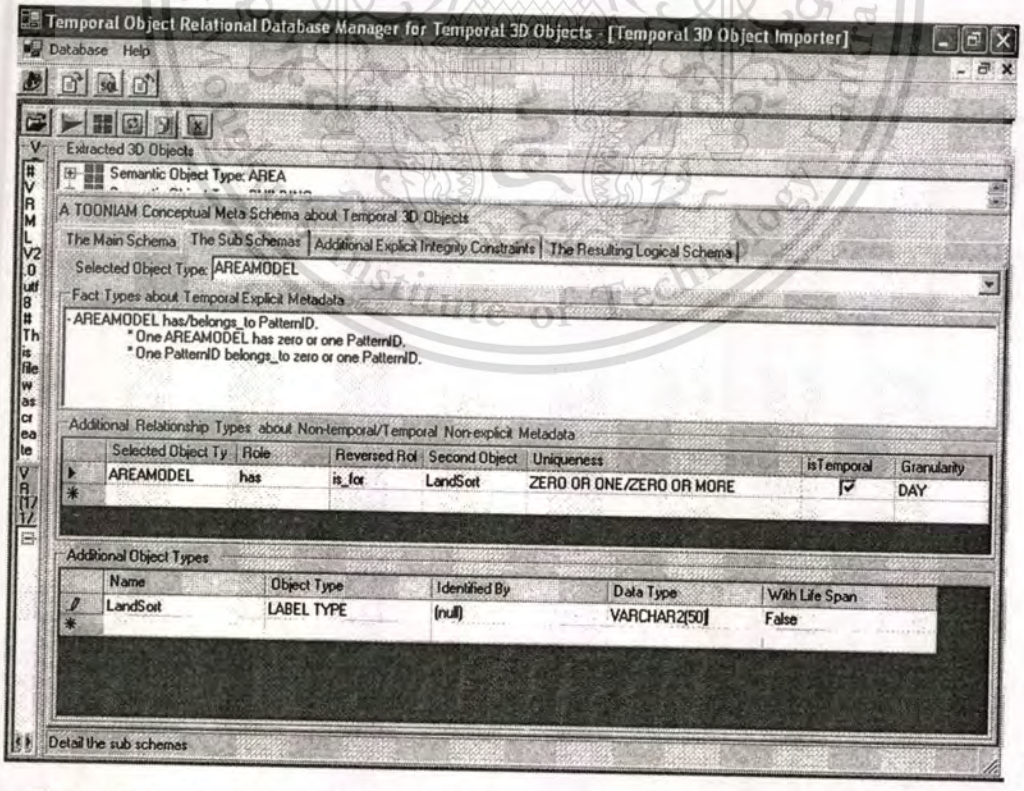


Figure B.11 The Detailing of the Sub Schema of the Category Type AreaModel

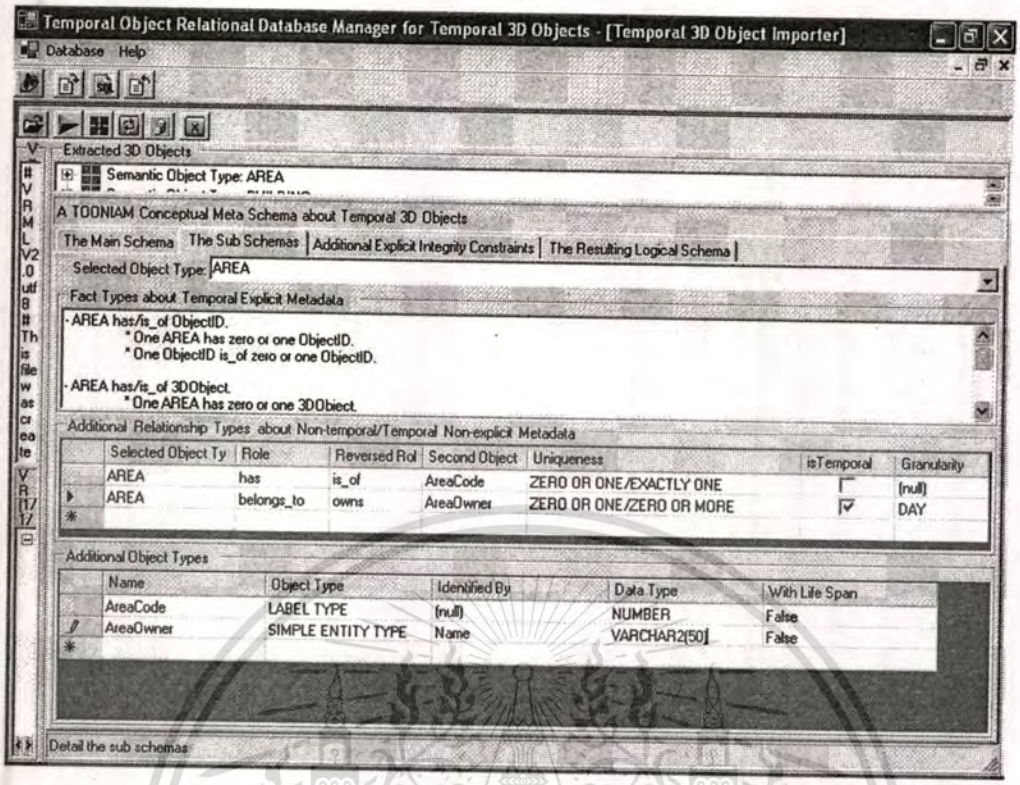


Figure B.12 The Detailing of the Sub Schema of the Temporal Complex Entity Type Area

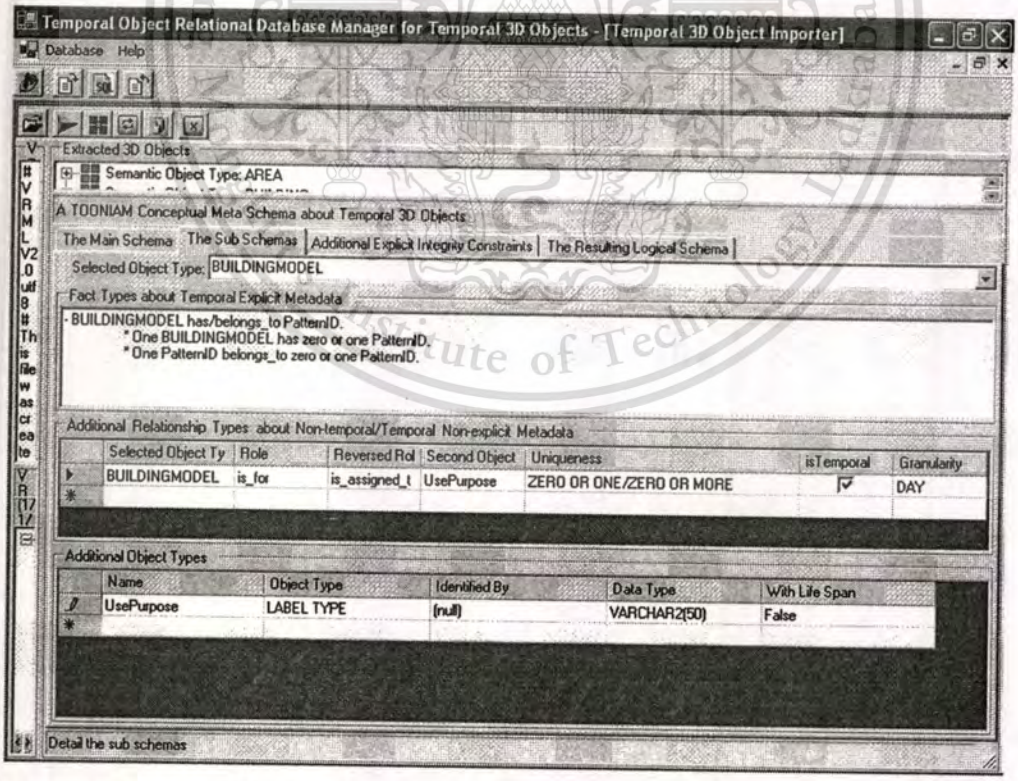


Figure B.13 The Detailing of the Sub Schema of the Category Type BuildingModel

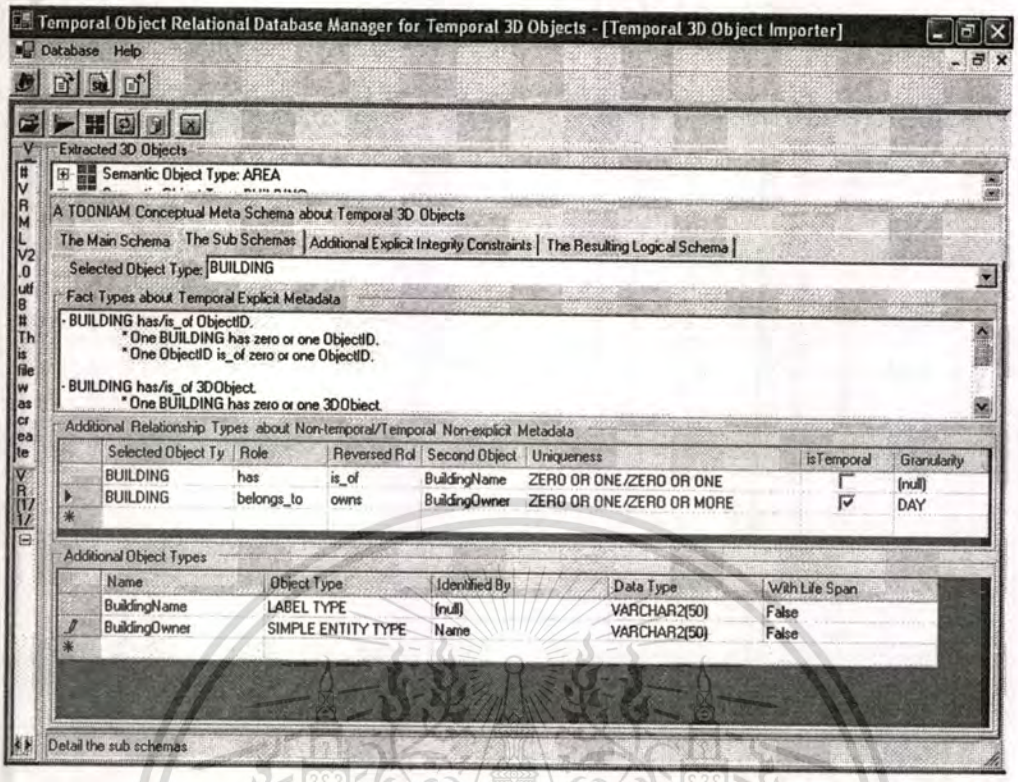


Figure B.14 The Detailing of the Sub Schema of the Temporal Complex Entity Type Building

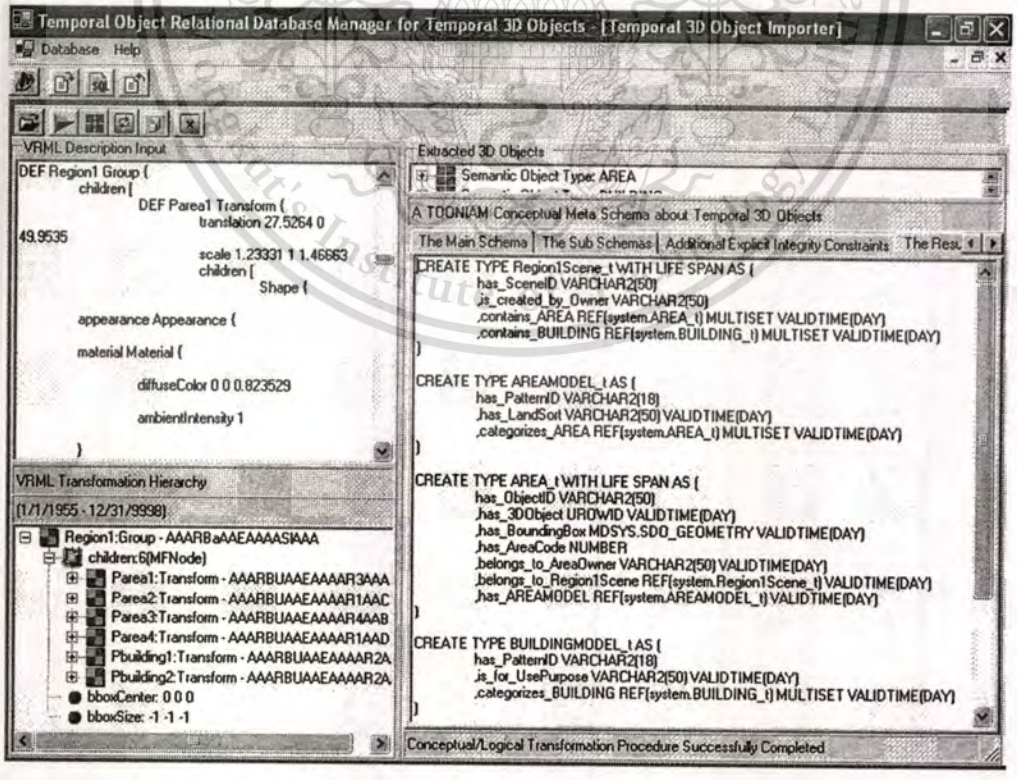


Figure B.15 The Obtained Logical Schema of the Temporal Object Relational Database for Temporal 3D Objects/Scenes

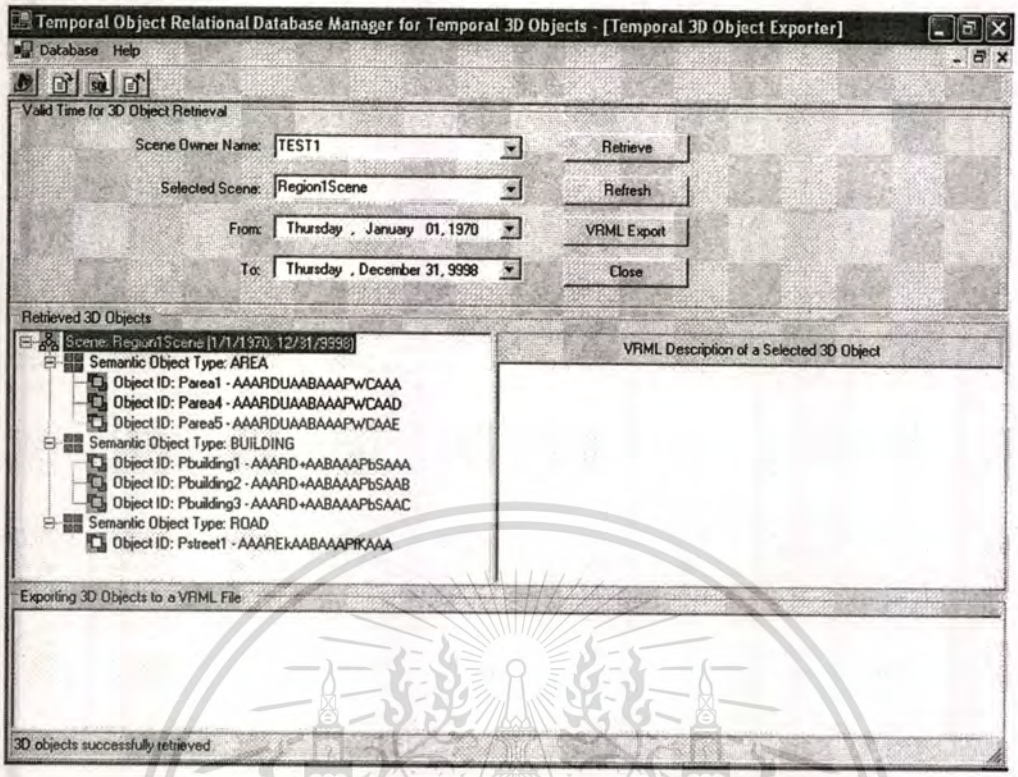


Figure B.16 Retrieved 3D Objects True for some Period of Time

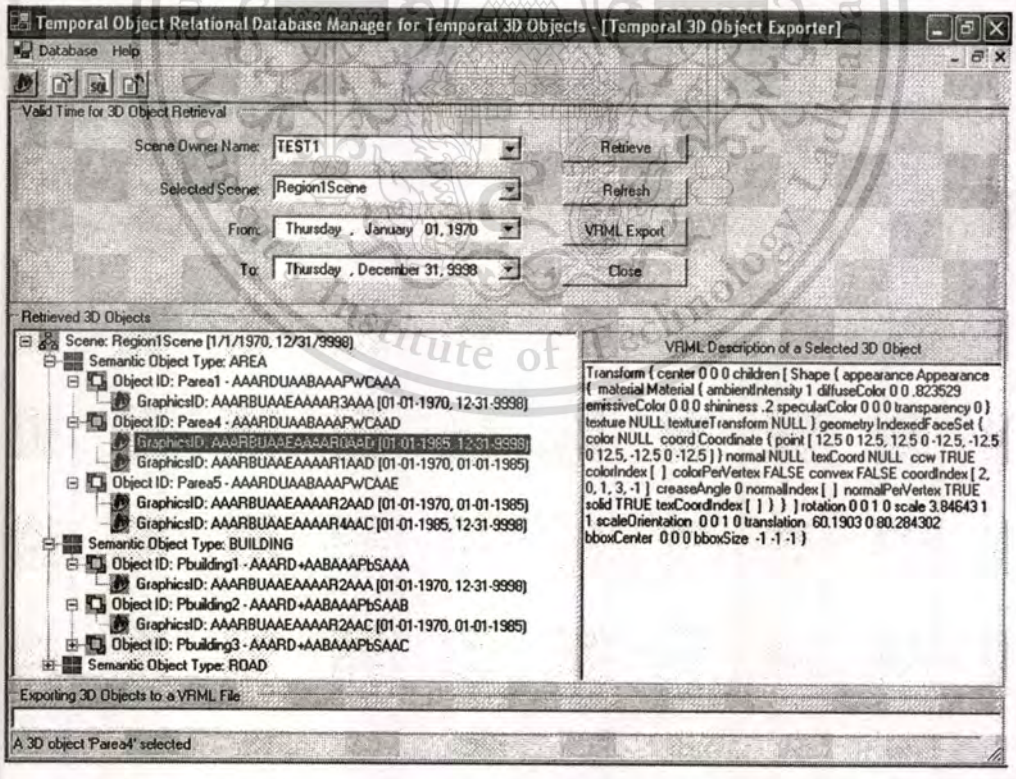


Figure B.17 The VRML Description of a Selected Retrieved 3D Object for some Period of Time

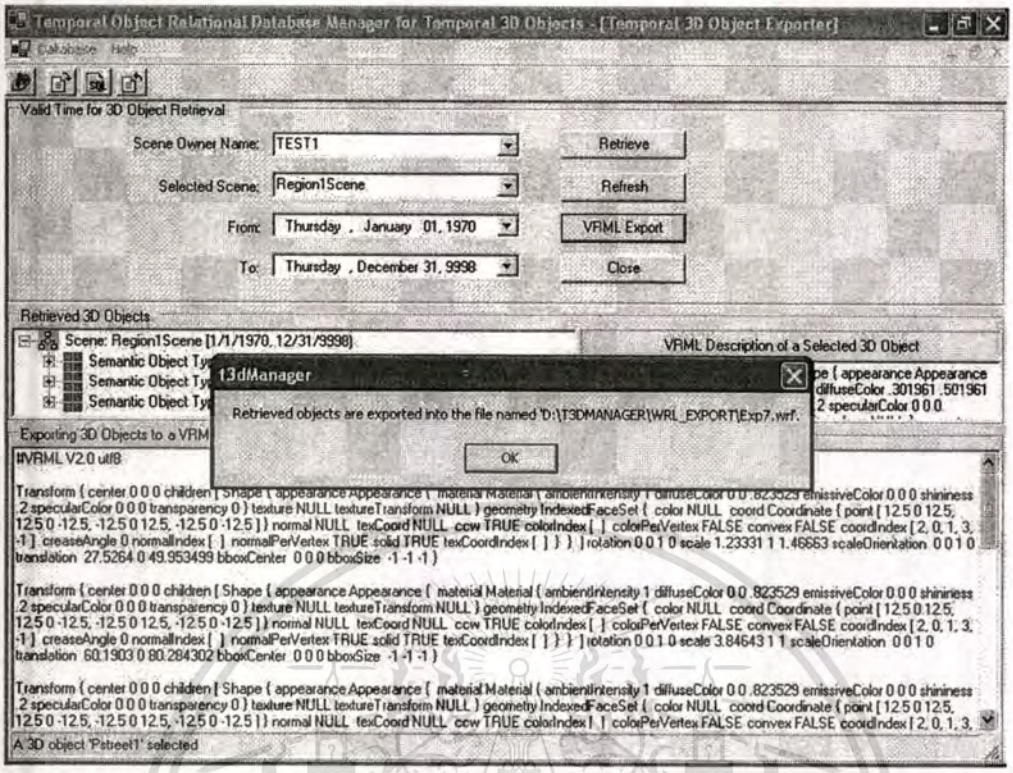


Figure B.18 The Exporting of the VRML Descriptions of all Retrieved 3D Objects for some Period of Time into the Exp7.wrl Target File

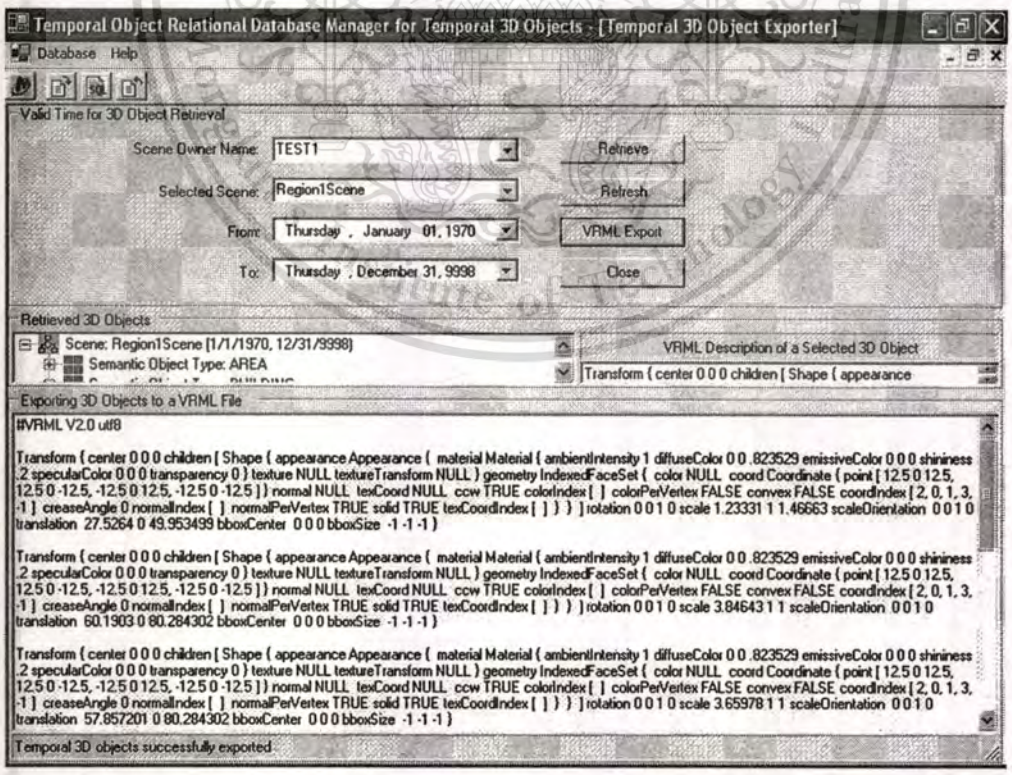


Figure B.19 The VRML Descriptions of the Retrieved 3D Objects for some Period of Time

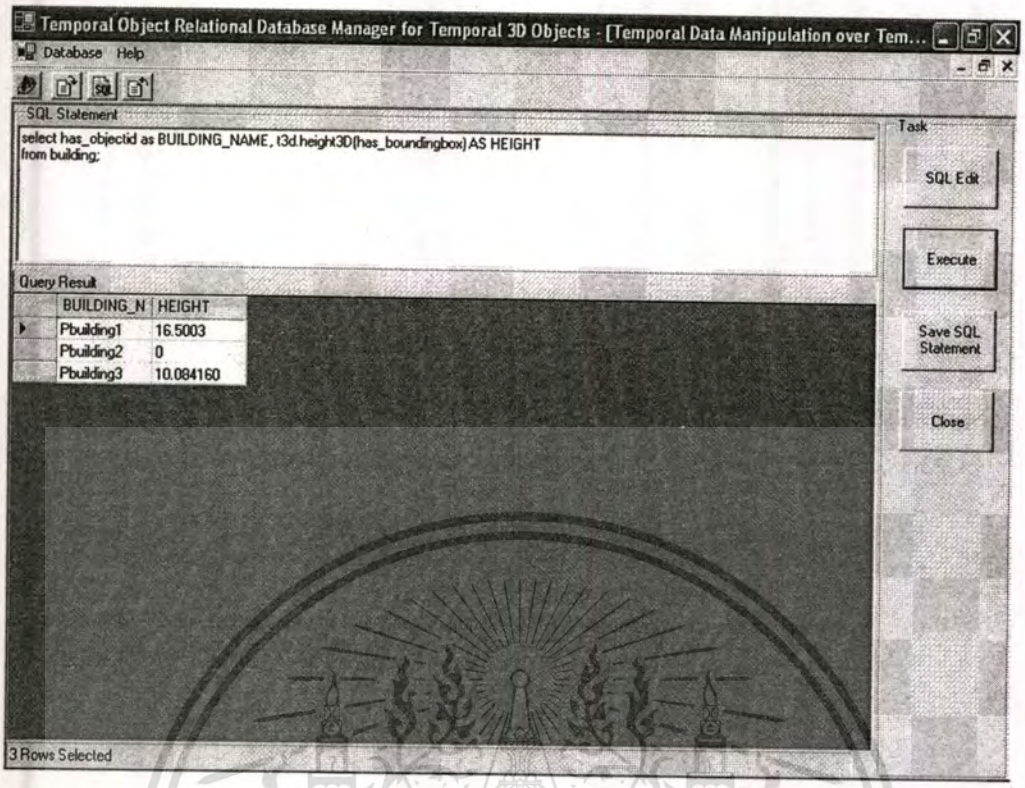


Figure B.20 A Non-temporal (Current) SELECT Statement on Explicit Metadata

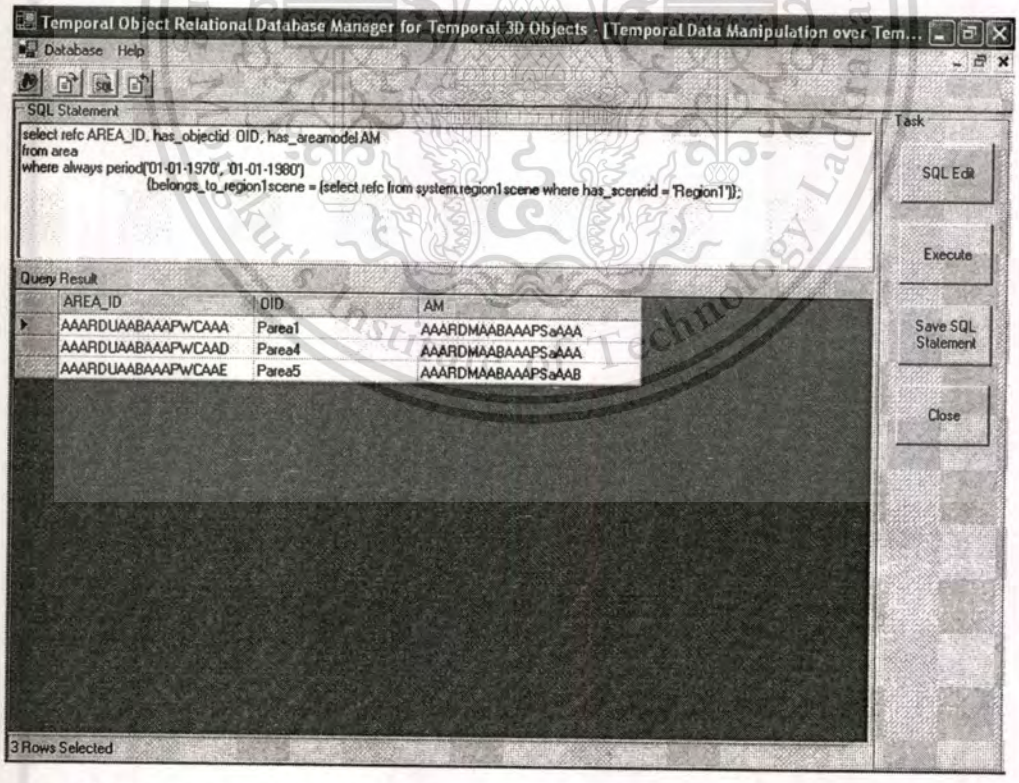


Figure B.21 A Temporal SELECT Statement on both Non-explicit and Explicit Metadata

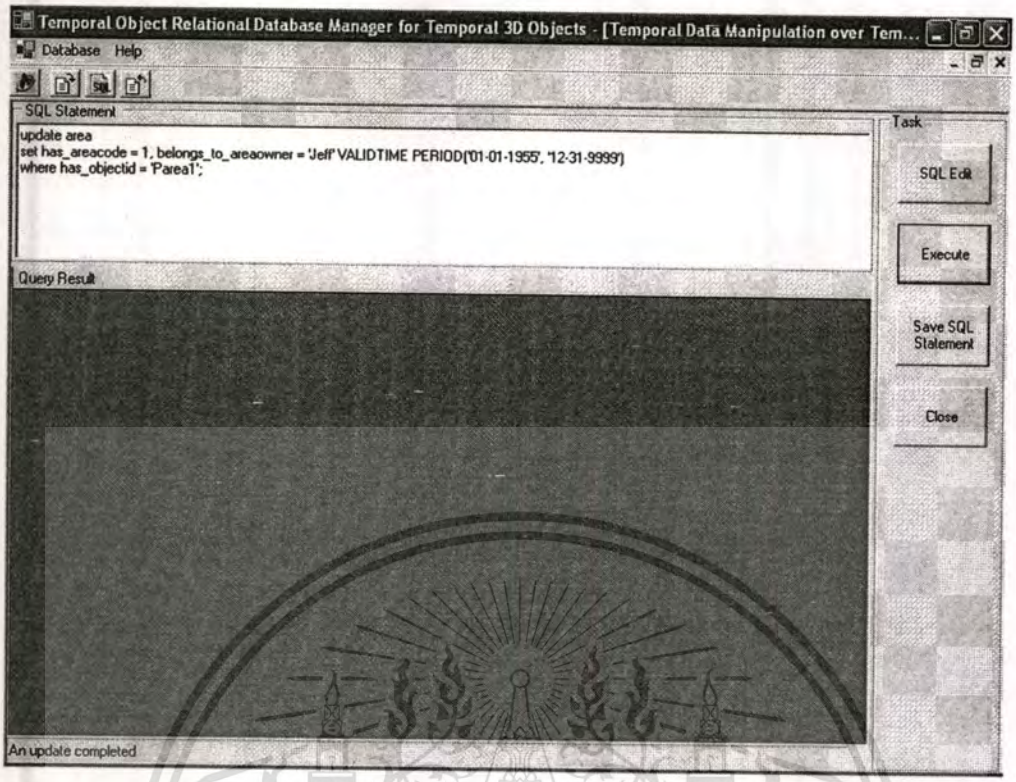


Figure B.22 A Temporal UPDATE Statement on Non-explicit Metadata

B.4 The SQL Statements of the Temporal Metadata Population Process

This subsection shows the SQL statements of the temporal metadata population process on each graphical source file of the case study in chapter 6.

B.4.1 For the Source File on 01/01/1955

CREATE TYPE system.Region1Scene_t WITH LIFE SPAN AS OBJECT (has_SceneID VARCHAR2(50), is_created_by_Owner VARCHAR2(50));

CREATE TYPE system.AREAMODEL_t AS OBJECT (has_PatternID VARCHAR2(18), has_LandSort VARCHAR2(50) VALIDTIME(DAY));

CREATE TYPE system.AREA_t WITH LIFE SPAN AS OBJECT (has_ObjectID VARCHAR2(50), has_3DObject UROWID VALIDTIME(DAY), has_BoundingBox MDSYS.SDO_GEOMETRY VALIDTIME(DAY), has_AreaCode NUMBER, belongs_to_AreaOwner VARCHAR2(50) VALIDTIME(DAY), belongs_to_Region1Scene REF(system.Region1Scene_t) VALIDTIME(DAY), has_AREAMODEL REF(system.AREAMODEL_t) VALIDTIME(DAY));

CREATE TYPE system.BUILDINGMODEL_t AS OBJECT (has_PatternID VARCHAR2(18), is_for_UsePurpose VARCHAR2(50) VALIDTIME(DAY));

CREATE TYPE system.BUILDING_t WITH LIFE SPAN AS OBJECT (has_ObjectID VARCHAR2(50), has_3DObject UROWID VALIDTIME(DAY), has_BoundingBox MDSYS.SDO_GEOMETRY VALIDTIME(DAY), has_BuildingName VARCHAR2(50), belongs_to_BuildingOwner VARCHAR2(50) VALIDTIME(DAY), belongs_to_Region1Scene REF(system.Region1Scene_t) VALIDTIME(DAY), has_BUILDINGMODEL REF(system.BUILDINGMODEL_t) VALIDTIME(DAY));

```
CREATE TABLE system.Region1Scene OF system.Region1Scene_t;

CREATE TABLE system.AREAMODEL OF system.AREAMODEL_t;

CREATE TABLE system.AREA OF system.AREA_t;

CREATE TABLE system.BUILDINGMODEL OF system.BUILDINGMODEL_t;

CREATE TABLE system.BUILDING OF system.BUILDING_t;

INSERT INTO system.Region1Scene(has_SceneID, is_created_by_Owner, LIFESPAN) VALUES
('Region1', NULL, T3D.VTPERIODS(T3D.VTPERIOD(TO_DATE('1-1-1955', 'mm-dd-yyyy'),
TO_DATE('12-31-9998', 'mm-dd-yyyy'))))

INSERT INTO system.AREAMODEL(has_PatternID, has_LandSort) SELECT
'AAAQ5ZAAEAAAAb/AAA', NULL FROM DUAL WHERE NOT EXISTS (SELECT * FROM
AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb/AAA');

INSERT INTO system.AREA(has_ObjectID, has_3DObject, has_BoundingBox, has_AreaCode,
belongs_to_AreaOwner, belongs_to_Region1Scene, has_AREAMODEL) VALUES ('Parea1',
'AAAQ4+AAEAAAAQQAAB' VALIDTIME PERIOD('1-1-1955', '12-31-9998'),
MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2),
SDO_ORDINATE_ARRAY(12.110025, 0, 31.620624, 42.942775, 0, 68.286374)) VALIDTIME
PERIOD('1-1-1955', '12-31-9998') , NULL, NULL, (SELECT ROWID FROM system.Region1Scene
WHERE has_SceneID = 'Region1') VALIDTIME PERIOD('1-1-1955', '12-31-9998'), (SELECT
ROWID FROM system.AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb/AAA')
VALIDTIME PERIOD('1-1-1955', '12-31-9998'));

INSERT INTO system.AREAMODEL(has_PatternID, has_LandSort) SELECT
'AAAQ5ZAAEAAAAb/AAA', NULL FROM DUAL WHERE NOT EXISTS (SELECT * FROM
AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb/AAA');

INSERT INTO system.AREA(has_ObjectID, has_3DObject, has_BoundingBox, has_AreaCode,
belongs_to_AreaOwner, belongs_to_Region1Scene, has_AREAMODEL) VALUES ('Parea2',
'AAAQ4+AAEAAAAQNAAB' VALIDTIME PERIOD('1-1-1955', '12-31-9998'),
MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2),
SDO_ORDINATE_ARRAY(42.440748, 0, 31.620624, 78.523248, 0, 68.286374)) VALIDTIME
PERIOD('1-1-1955', '12-31-9998') , NULL, NULL, (SELECT ROWID FROM system.Region1Scene
WHERE has_SceneID = 'Region1') VALIDTIME PERIOD('1-1-1955', '12-31-9998'), (SELECT
ROWID FROM system.AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb/AAA')
VALIDTIME PERIOD('1-1-1955', '12-31-9998'));

INSERT INTO system.AREAMODEL(has_PatternID, has_LandSort) SELECT
'AAAQ5ZAAEAAAAcAAAA', NULL FROM DUAL WHERE NOT EXISTS (SELECT * FROM
AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAcAAAA');

INSERT INTO system.AREA(has_ObjectID, has_3DObject, has_BoundingBox, has_AreaCode,
belongs_to_AreaOwner, belongs_to_Region1Scene, has_AREAMODEL) VALUES ('Parea3',
'AAAQ4+AAEAAAAQPAAC' VALIDTIME PERIOD('1-1-1955', '12-31-9998'),
MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2),
SDO_ORDINATE_ARRAY(78.021073, 0, 31.620624, 103.604323, 0, 68.286374)) VALIDTIME
PERIOD('1-1-1955', '12-31-9998') , NULL, NULL, (SELECT ROWID FROM system.Region1Scene
WHERE has_SceneID = 'Region1') VALIDTIME PERIOD('1-1-1955', '12-31-9998'), (SELECT
ROWID FROM system.AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAcAAAA')
VALIDTIME PERIOD('1-1-1955', '12-31-9998'));

INSERT INTO system.AREAMODEL(has_PatternID, has_LandSort) SELECT
'AAAQ5ZAAEAAAAb/AAA', NULL FROM DUAL WHERE NOT EXISTS (SELECT * FROM
AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb/AAA');
```

INSERT INTO system.AREA(has_ObjectID, has_3DObject, has_BoundingBox, has_AreaCode, belongs_to_AreaOwner, belongs_to_Region1Scene, has_AREAMODEL) VALUES ('Parea4', 'AAAQ4+AAEAAAAQMAAB' VALIDTIME PERIOD('1-1-1955', '12-31-9998'), MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.109951, 0, 67.784302, 103.604451, 0, 92.784302)) VALIDTIME PERIOD('1-1-1955', '12-31-9998') , NULL, NULL, (SELECT ROWID FROM system.Region1Scene WHERE has_SceneID = 'Region1') VALIDTIME PERIOD('1-1-1955', '12-31-9998'), (SELECT ROWID FROM system.AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb/AAA') VALIDTIME PERIOD('1-1-1955', '12-31-9998'));

INSERT INTO system.BUILDINGMODEL(has_PatternID, is_for_UsePurpose) SELECT 'AAAQ5ZAAEAAAAb+AAA', NULL FROM DUAL WHERE NOT EXISTS (SELECT * FROM BUILDINGMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb+AAA');

INSERT INTO system.BUILDING(has_ObjectID, has_3DObject, has_BoundingBox, has_BuildingName, belongs_to_BuildingOwner, belongs_to_Region1Scene, has_BUILDINGMODEL) VALUES ('Pbuilding1', 'AAAQ4+AAEAAAAQPAAD' VALIDTIME PERIOD('1-1-1955', '12-31-9998'), MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(19.54722, 0, 38.2631, 39.5539774432, 16.5003, 48.2631)) VALIDTIME PERIOD('1-1-1955', '12-31-9998') , NULL, NULL, (SELECT ROWID FROM system.Region1Scene WHERE has_SceneID = 'Region1') VALIDTIME PERIOD('1-1-1955', '12-31-9998'), (SELECT ROWID FROM system.BUILDINGMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb+AAA') VALIDTIME PERIOD('1-1-1955', '12-31-9998'));

INSERT INTO system.BUILDINGMODEL(has_PatternID, is_for_UsePurpose) SELECT 'AAAQ5ZAAEAAAAb9AAA', NULL FROM DUAL WHERE NOT EXISTS (SELECT * FROM BUILDINGMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb9AAA');

INSERT INTO system.BUILDING(has_ObjectID, has_3DObject, has_BoundingBox, has_BuildingName, belongs_to_BuildingOwner, belongs_to_Region1Scene, has_BUILDINGMODEL) VALUES ('Pbuilding2', 'AAAQ4+AAEAAAAQNAAC' VALIDTIME PERIOD('1-1-1955', '12-31-9998'), MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(58.523478, 0, 37.954102, 71.189718, 13.00058, 56.120102)) VALIDTIME PERIOD('1-1-1955', '12-31-9998') , NULL, NULL, (SELECT ROWID FROM system.Region1Scene WHERE has_SceneID = 'Region1') VALIDTIME PERIOD('1-1-1955', '12-31-9998'), (SELECT ROWID FROM system.BUILDINGMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb9AAA') VALIDTIME PERIOD('1-1-1955', '12-31-9998'));

B.4.2 For the Source File on 01/01/1965

UPDATE Region1Scene SET LIFESPAN = T3D.TSQL.tUNION(LIFESPAN, T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1965', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy')))) WHERE has_SceneID = 'Region1'

UPDATE AREA SET LIFESPAN = T3D.TSQL.tUNION(LIFESPAN, T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1965', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy')))) WHERE has_ObjectID = 'Parea1'

UPDATE system.AREA SET has_AREAMODEL = (SELECT ROWID FROM system.AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb/AAA') VALIDTIME PERIOD('1-1-1965', '12-31-9998') WHERE has_ObjectID = 'Parea1';

UPDATE system.AREA SET has_3DObject = 'AAAQ4+AAEAAAAQQAAB' VALIDTIME PERIOD('1-1-1965', '12-31-9998') WHERE has_ObjectID = 'Parea1';

UPDATE system.AREA SET has_BoundingBox = MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.110025, 0, 31.620624, 42.942775, 0, 68.286374)) VALIDTIME PERIOD('1-1-1965', '12-31-9998') WHERE has_ObjectID = 'Parea1';

UPDATE AREA SET LIFESPAN = T3D.TSQL.tUNION(LIFESPAN, T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1965', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy')))) WHERE has_ObjectID = 'Parea4'

UPDATE system.AREA SET has_AREAMODEL = (SELECT ROWID FROM system.AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb/AAA') VALIDTIME PERIOD('1-1-1965', '12-31-9998') WHERE has_ObjectID = 'Parea4';

UPDATE system.AREA SET has_3DObject = 'AAAQ4+AAEAAAAQMAAB' VALIDTIME PERIOD('1-1-1965', '12-31-9998') WHERE has_ObjectID = 'Parea4';

UPDATE system.AREA SET has_BoundingBox = MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.109951, 0, 67.784302, 103.604451, 0, 92.784302)) VALIDTIME PERIOD('1-1-1965', '12-31-9998') WHERE has_ObjectID = 'Parea4';

INSERT INTO system.AREAMODEL (HAS_PATTERNID, HAS_LANDSORT) SELECT 'AAAQ5ZAAEAAAAcAAAA', NULL FROM DUAL WHERE NOT EXISTS (SELECT * FROM system.AREAMODEL WHERE HAS_PATTERNID = 'AAAQ5ZAAEAAAAcAAAA')

INSERT INTO system.AREA(has_ObjectID, has_3DObject, has_BoundingBox, HAS_AREACODE, BELONGS_TO_AREAOWNER, belongs_to_Region1Scene, has_AREAMODEL) VALUES ('Parea5', 'AAAQ4+AAEAAAAQNAAD' VALIDTIME PERIOD('1-1-1965', '12-31-9998'), MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(42.440848, 0, 31.620624, 103.604348, 0, 68.286374)) VALIDTIME PERIOD('1-1-1965', '12-31-9998') , NULL, NULL, (SELECT ROWID FROM system.Region1Scene WHERE has_SceneID = 'Region1') VALIDTIME PERIOD('1-1-1965', '12-31-9998'), (SELECT ROWID FROM system.AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAcAAAA') VALIDTIME PERIOD('1-1-1965', '12-31-9998'));

UPDATE AREA SET LIFESPAN = T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1965', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy'))) WHERE has_ObjectID = 'Parea5'

UPDATE BUILDING SET LIFESPAN = T3D.TSQL.tUNION(LIFESPAN, T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1965', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy')))) WHERE has_ObjectID = 'Pbuilding1'

UPDATE system.BUILDING SET has_BUILDINGMODEL = (SELECT ROWID FROM system.BUILDINGMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb+AAA') VALIDTIME PERIOD('1-1-1965', '12-31-9998') WHERE has_ObjectID = 'Pbuilding1';

UPDATE system.BUILDING SET has_3DObject = 'AAAQ4+AAEAAAAQPAAD' VALIDTIME PERIOD('1-1-1965', '12-31-9998') WHERE has_ObjectID = 'Pbuilding1';

UPDATE system.BUILDING SET has_BoundingBox = MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(19.54722, 0, 38.2631, 39.5539774432, 16.5003, 48.2631)) VALIDTIME PERIOD('1-1-1965', '12-31-9998') WHERE has_ObjectID = 'Pbuilding1';

UPDATE BUILDING SET LIFESPAN = T3D.TSQL.tUNION(LIFESPAN, T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1965', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy')))) WHERE has_ObjectID = 'Pbuilding2'

UPDATE system.BUILDING SET has_BUILDINGMODEL = (SELECT ROWID FROM system.BUILDINGMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb9AAA') VALIDTIME PERIOD('1-1-1965', '12-31-9998') WHERE has_ObjectID = 'Pbuilding2';

UPDATE system.BUILDING SET has_3DObject = 'AAAQ4+AAEAAAAQNAAC' VALIDTIME PERIOD('1-1-1965', '12-31-9998') WHERE has_ObjectID = 'Pbuilding2';

```
UPDATE system.BUILDING SET has_BoundingBox = MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(58.523478, 0, 37.954102, 71.189718, 13.00058, 56.120102)) VALIDTIME PERIOD('1-1-1965', '12-31-9998') WHERE has_ObjectID = 'Pbuilding2';

CREATE TYPE system.ROADMODEL_t AS OBJECT (has_PatternID VARCHAR2(18), is_for_VehicleKind VARCHAR2(50));

CREATE TYPE system.ROAD_t WITH LIFE SPAN AS OBJECT (has_ObjectID VARCHAR2(50), has_3DObject UROWID VALIDTIME(DAY), has_BoundingBox MDSYS.SDO_GEOMETRY VALIDTIME(DAY), has_RoadName VARCHAR2(50), belongs_to_Region1Scene REF(system.Region1Scene_t) VALIDTIME(DAY), has_ROADMODEL REF(system.ROADMODEL_t) VALIDTIME(DAY));

CREATE TABLE system.ROADMODEL OF system.ROADMODEL_t;

CREATE TABLE system.ROAD OF system.ROAD_t;

INSERT INTO system.ROADMODEL(has_PatternID, is_for_VehicleKind) SELECT 'AAAQ5ZAAEAAAAb8AAA', NULL FROM DUAL WHERE NOT EXISTS (SELECT * FROM system.ROADMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb8AAA');

INSERT INTO system.ROAD(has_ObjectID, has_3DObject, has_BoundingBox, has_RoadName, belongs_to_Region1Scene, has_ROADMODEL) VALUES ('Pstreet1', 'AAAQ4+AAEAAAAQQAAC' VALIDTIME PERIOD('1-1-1965', '12-31-9998'), MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.1973014272, 0, 28.208861, 103.5171005728, 0, 32.034941)) VALIDTIME PERIOD('1-1-1965', '12-31-9998') , NULL, (SELECT ROWID FROM system.Region1Scene WHERE has_SceneID = 'Region1') VALIDTIME PERIOD('1-1-1965', '12-31-9998'), (SELECT ROWID FROM system.ROADMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb8AAA') VALIDTIME PERIOD('1-1-1965', '12-31-9998'));

UPDATE system.ROAD SET LIFESPAN = T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1965', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy'))) WHERE has_ObjectID = 'Pstreet1';
```

B.4.3 For the Source File on 01/01/1985

```
UPDATE Region1Scene SET LIFESPAN = T3D.TSQL.tUNION(LIFESPAN, T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1985', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy')))) WHERE has_SceneID = 'Region1'

UPDATE AREA SET LIFESPAN = T3D.TSQL.tUNION(LIFESPAN, T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1985', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy')))) WHERE has_ObjectID = 'Parea1'

UPDATE system.AREA SET has_AREAMODEL = (SELECT ROWID FROM system.AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb/AAA') VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Parea1';

UPDATE system.AREA SET has_3DObject = 'AAAQ4+AAEAAAAQQAAB' VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Parea1';

UPDATE system.AREA SET has_BoundingBox = MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.110025, 0, 31.620624, 42.942775, 0, 68.286374)) VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Parea1';

UPDATE AREA SET LIFESPAN = T3D.TSQL.tUNION(LIFESPAN, T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1985', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy')))) WHERE has_ObjectID = 'Parea4'
```

```
UPDATE system.AREA SET has_AREAMODEL = (SELECT ROWID FROM system.AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb/AAA') VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Parea4';

UPDATE system.AREA SET has_3DObject = 'AAAQ4+AAEAAAAQMAAC' VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Parea4';

UPDATE system.AREA SET has_BoundingBox = MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.109925, 0, 67.784302, 108.270675, 0, 92.784302)) VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Parea4';

UPDATE AREA SET LIFESPAN = T3D.TSQL.tUNION(LIFESPAN, T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1985', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy')))) WHERE has_ObjectID = 'Parea5'

UPDATE system.AREA SET has_AREAMODEL = (SELECT ROWID FROM system.AREAMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAcAAAA') VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Parea5';

UPDATE system.AREA SET has_3DObject = 'AAAQ4+AAEAAAAQQAAD' VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Parea5';

UPDATE system.AREA SET has_BoundingBox = MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(42.440698, 0, 31.620624, 108.270698, 0, 68.286374)) VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Parea5';

UPDATE ROAD SET LIFESPAN = T3D.TSQL.tUNION(LIFESPAN, T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1985', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy')))) WHERE has_ObjectID = 'Pstreet1'

UPDATE system.ROAD SET has_ROADMODEL = (SELECT ROWID FROM system.ROADMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb8AAA') VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Pstreet1';

UPDATE system.ROAD SET has_3DObject = 'AAAQ4+AAEAAAAQQAAC' VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Pstreet1';

UPDATE system.ROAD SET has_BoundingBox = MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(12.1973014272, 0, 28.208861, 103.5171005728, 0, 32.034941)) VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Pstreet1';

INSERT INTO system.BUILDINGMODEL (HAS_PATTERNID, IS_FOR_USEPURPOSE) SELECT 'AAAQ5ZAAEAAAAb9AAA', NULL FROM DUAL WHERE NOT EXISTS (SELECT * FROM system.BUILDINGMODEL WHERE HAS_PATTERNID = 'AAAQ5ZAAEAAAAb9AAA')

INSERT INTO system.BUILDING(has_ObjectID, has_3DObject, has_BoundingBox, HAS_BUILDINGNAME, BELONGS_TO_BUILDINGOWNER, belongs_to_Region1Scene, has_BUILDINGMODEL) VALUES ('Pbuilding3', 'AAAQ4+AAEAAAAQNAAE' VALIDTIME PERIOD('1-1-1985', '12-31-9998'), MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(32.275639, 0, 76.450897, 61.857159, 10.08416, 86.450897)) VALIDTIME PERIOD('1-1-1985', '12-31-9998') , NULL, NULL, (SELECT ROWID FROM system.Region1Scene WHERE has_SceneID = 'Region1') VALIDTIME PERIOD('1-1-1985', '12-31-9998'), (SELECT ROWID FROM system.BUILDINGMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb9AAA') VALIDTIME PERIOD('1-1-1985', '12-31-9998'));

UPDATE BUILDING SET LIFESPAN = T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1985', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy'))) WHERE has_ObjectID = 'Pbuilding3'
```

```
UPDATE BUILDING SET LIFESPAN = T3D.TSQL.tUNION(LIFESPAN, T3D.VTPERIODS(t3D.VTPERIOD(TO_DATE('1-1-1985', 'mm-dd-yyyy'), TO_DATE('12-31-9998', 'mm-dd-yyyy')))) WHERE has_ObjectID = 'Pbuilding1'

UPDATE system.BUILDING SET has_BUILDINGMODEL = (SELECT ROWID FROM system.BUILDINGMODEL WHERE has_PatternID = 'AAAQ5ZAAEAAAAb+AAA') VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Pbuilding1';

UPDATE system.BUILDING SET has_3DObject = 'AAAQ4+AAEAAAAQPAAD' VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Pbuilding1';

UPDATE system.BUILDING SET has_BoundingBox = MDSYS.SDO_GEOMETRY(3005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 2), SDO_ORDINATE_ARRAY(19.54722, 0, 38.2631, 39.5539774432, 16.5003, 48.2631)) VALIDTIME PERIOD('1-1-1985', '12-31-9998') WHERE has_ObjectID = 'Pbuilding1';
```

Author Biography

Vo Thi Ngoc Chau received B.Eng. and M.Eng. degrees in Computer Engineering from Ho Chi Minh City University of Technology, Vietnam, and King Mongkut's Institute of Technology Ladkrabang (KMITL), Thailand, in 2003 and 2005, respectively. After that, she continued the Ph.D. study at KMITL.

The research work of her Master's and Doctoral studies has been carried out under the Southeast Asia Engineering Education Development Network (SEED-Net) Project of the ASEAN University Network (AUN).

Her research of interest focuses on temporal databases, temporal data warehouses, and data modeling.