Evolving Document Type Definitions (DTDs) for Integrated Health Data.

Tore Mallaug[1, 2], Kjell Bratbergsengen[1]
torem@aitel.hist.no, kjellb@idi.ntnu.no

[1]The Department of Computer and Information Science,
Faculty of Information Technology, Mathematics and Electrical Engineering,
Norwegian University of Science and Technology (NTNU)
[2]Faculty of Informatics and e-Learning, Sør-Trønderlag University College
(Høgskolen i Sør-Trønderlag)

ABSTRACT

The appetite for timely, accurate health data is continuously growing (e.g. [DiSt97]). The increasing volume of data collected from servers and other sources creates new needs for a future totally integrated health information system. The database (structure and system) plays a very important role in such a future system. All kinds of structured and semi-structured personal health data must be stored and represented for long-term use. For this purpose a simple temporal object model is introduced. Any stored objects are given time stamps on a linear time axis. Temporal / historical perspectives on long-term stored data may be generated by searching these time stamps.

To represent the data objects and their metadata XML and its Document Type Definitions (DTDs) is suggested as an implementation solution.

Keywords

Database, Data model, Temporal data model, Electronic health data, Computer-based patient record system (CPR), XML, Document Type Definitions (DTDs), Semi structured data, Long-term storage.

1. INTRODUCTION

1.1 Introduction

In a future totally integrated health information system (simplified called TIHS) citizens will be able to control and browse their whole individual electronic health record (EHR) from a single source. The database (structure and system) plays a crucial role in such a future system. All kinds of personal health data, including semi structured data, must be stored and represented for long-term use. For this purpose a simple temporal object model is introduced. Any stored objects in the database are given time stamps on a linear time axis. Temporal / historical perspectives on long-term stored data may be generated by searching these time stamps. By using a database concept based on a log-only principle, stored objects are never changed or deleted. Any update of data results in new object versions. Each version is reflecting it's time: (medical) terminology, procedures, development, practice, needs, and functionality. To never delete objects give a unique possibility to re-construct any past times of selected periods / intervals. The suggested

approach represents nearly unlimited ways for applications to retrieve (query) and view historical data from the database.

To represent the data objects and their metadata XML (Extensible Markup Language) [XML] and its Document Type Definitions (DTDs) is suggested as an implementation solution.

1.2 Motivation

The increasing volume of health related data creates new needs for future extended computer-based patient record systems (CPR's -e.g. [DiSt97][BeMu97]). A higher level of the CPR may be called an electronic health record (EHR)[Waeg96]. In an EHR recorded data are not limited to traditional patient medical treatments only, but include a wider term of wellness and material data, like employment and socio-economic data. The future EHR should be available on demand independently of where the medical treatment and other usage are taking place. Other usage includes medical research (both retrospective and prospective [ShBa90]), logging of use, medical education and health statistics (e.g. purposes in health care listed by [BiBl94]). In the future, individuals can share their own health data according to their need for health care. Citizens can 'carry' their own EHR to the service providers, and control the access and use of their registered data. A common integrated database replaces any future need for smart card solutions for this data 'carry'. However, data security mechanisms needed for accessing the EHR, and harmonizing to data security laws are an important and separate topic not discussed in this paper. Stored health data should be correct, legal, updated, available, useful and secured against abuse. The choice of database solutions and storage strategies should be a part of the work to reach these essential goals in a future TIHS. Database solutions affect the total information system, the storage cost and management, security and data communication.

1.3 XML and DTD

XML is rapidly emerging as a standard for exchanging data. Its nested, self-describing structure provides a simple yet flexible means for applications to exchange data. Document Type Definitions (DTDs) [DTD98] are essentially schemas for XML documents. DTDs are being developed for a range of domains, like several industry proposals.

In this paper the term DTD is used very generally for a consistent, formal and structured way to define semi-structured documents as objects including metadata elements for long-term storage. DTDs lack some useful database schema functionalities, like the possibility to define relationships between instances in the database. XML Schema [XMLS01] gives a better schema facility. XML Schema may be seen as an extension of DTD, as a one-to-one relationship from DTD to XML Schema. The proposed use of evolving DTDs in this paper is so general that it is supposed to work for XML Schemas or any future schema extensions as well.

1.4 A temporal perspective

Information retrieval and reuse becomes more and more important both for medical treatment and research. The treatment is moving to a more process oriented and seamless health care delivery system, where the treatment is a link of different actions done by usually geographically distant actors (e.g. [HCIN00]). Such processes are also typical periodically, like they are going on for a longer or shorter period in a person's life (e.g.

[KITH00]). Or the same health problem can return periodically during the lifetime of an individual. The representation of the whole integrated patient history in a temporal data model makes it possible to follow links in time for examination the course of events whatever where, when and by whom the involved parts of a total treatment are registered in the EHR. This temporal overview of a persons health history can be useful in future treatments, research and statistics, and also as a documentation for possible legal actions later on.

In modern research, like in genome research, the need to have chronological historical data available is essential for studies of possible alternative courses of events (e.g. [PaGo01]). A collection-driven "hunting" into collected data is often desirable rather than doing hypothesis-driven research.

By implementing ONE logically common database, the stored data elements can easily be linked to a common temporal data model where all registered data is represented as temporal data objects arranged in order of the valid time they where registered in the system.

As far as we know, no common database solution strategy exists in the world of health informatics today. [Elvira01] suggests distributed system solutions for collection and visualization of patient data from several sources, but gives no details about any common database. The status of today is several (local) heterogeneous information systems that are more or less not able to communicate. During the last 10 years [Bakk00] actions has been done in Norway trying to agree on a common massage standard for exchanging data between health informatics systems. However, there is still no standard covering all kinds of contents and message types.

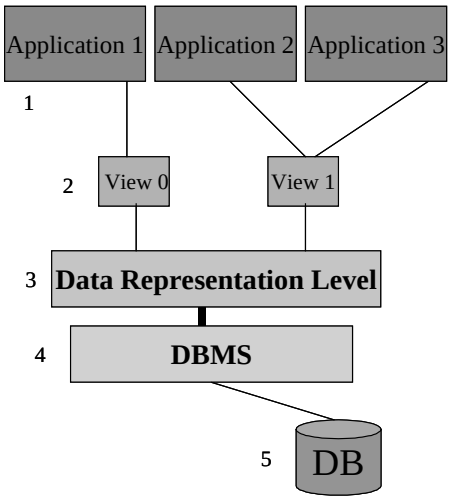


Figure 1: Parts in a Total Integrated Health System (THIS)

In the rest of the paper we introduce shortly our thought about a TIHS and a temporal data model in section 2. Section 3 shows how applications can interact the database by accessing the temporal data representation level. Section 4 summarizes and points out some topics for future work.

2. A Total Integrated Health System (TIHS)

2.1 Overview

The total integrated health system (TIHS) includes the following parts (numbers refer levels in figure 1) and the data transfer between them (the lines in the figure):

1. Heterogeneous local applications
2. A set of views

3. A common data representation level (DRL)
4. The Database system (DBMS)
5. A Physical storage of the database

All data connected to the EHR and the usage of EHR is a part of the TIHS. However, local administration systems and their business data like financial and administrative hospital data about inpatients, is not a scope in our definition of TIHS. We concentrate on all kind of person-oriented data, information and knowledge that may be addressed as health data in any form. The system in figure 1 works as follows (numbers in parenthesis refer to the numbers in the figure):

Local applications (1), or information systems, interact with the common database through the views (2). Applications could be CPR (as earlier mentioned) including multimedia options like system for representing x-ray pictures, e-health applications, wireless laptops at hospitals, and terminals for public use by citizens. A view works as a link between the application and a common data representation level abbreviated to DRL (3). How views are to be implemented, and how many views to have are not discussed in this paper. The DRL has several functions. The main reason to implement a DRL is to create a common environment for our temporal data model, storing versions of data objects and schemas in a temporal space (described in the next subsection). DRL makes it possible to represent data from the local applications as versions of objects in the temporal space. These versions can be linked together by including mapping rules in DRL. Since DRL gives a common representation of any data objects, including metadata, this implies that communication between local applications become easier. Instead of sending messages between applications, using emails or EDI, any applications can just access data from the common DRL (and indirectly from the common database) if they have the access rights to do so. Instead of message exchange standardization between local systems, we rather suggest standard representations of any stored health data. We have put representations in plural, since in a very long-term perspective we probably have to cope with more than one standard. Several schemas will show up over time, some semantically richer than others. A common DRL makes it easier to create mappings, or converting rules, between old and new schemas. Today, according to the fact that this enormous amount of data will be semi-structural, the XML standard may be a good choice, including its use of DTD and XML Schema. However, in a long-term perspective it's hard to know how long XML will survive, so XML works more like an example here. In an XML/DTD context, each object type may be defined by a DTD, and each data instance of an object type is linked and validated to known DTD versions stored in the TIHS.

As an additional feature, DRL works like a middleware to the common DBMS (4). This means that local applications don't need to know how the DBMS is implemented or where the data is physically stored (5). Each application only needs to have a link to a given view that works. A good implementation of DRL replaces the needs for other middleware solutions.

Here we say nothing about how the common DBMS is implemented. It's important to note that the DBMS works independently of DRL. The only requirement is that the DBMS must

be able to store all the objects represented in DRL. In fact, the DBMS does not need to be implemented as a temporal database at all. It's the DRL that include the temporal aspect to the data, the DBMS can simply store data according to a given implementation data model, like the relational data model.

2.2 The Temporal Object Model

A simple temporal object model is introduced. The model accepts versions of data objects; each connected to a corresponding DTD version. DTDs are represented and stored as own objects as well. All stored objects, including DTD objects, must be related to a set of time stamps. This set includes a fixed time stamp called *object date* that is the transaction time of when the new object instance was created/inserted into the database. A given object version instance is identified by its object date and a universal unique object identifier (OID).

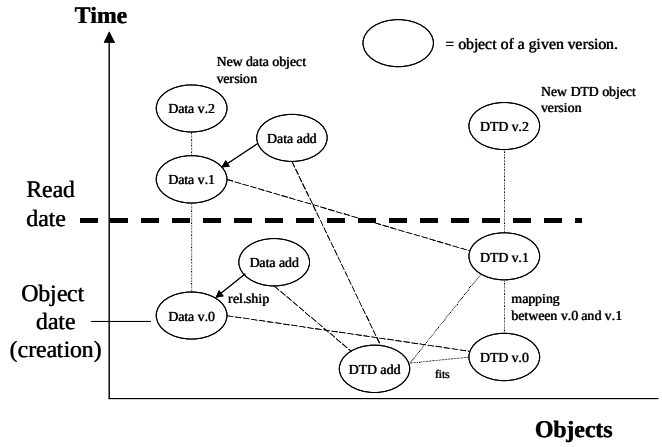


Figure 2: Versions of Data objects and DTD objects by time.

In addition, other types of time stamps may be associated to an instance (from now on called an object).

Transaction time and valid time might differ. This may be the case if an object represents an observation that is not immediately stored in the database. In the future it might be that the any hospital's staff may type information directly into the system by using, say, wireless laptops. But there might still be delays created by software (like how the programs are moving the data to the database), hardware (like delays in data communication) or practical (like work flow issues) reasons. However, the valid time may be included in an object as an additional time stamp. Then the users, the applications, may decide by themselves later on which time stamp they like to access. Another example is how to associate an object to historical times, like if the object represent corrected or added data to previous object versions. Additional time stamps may as well be introduced here for representing links back in time, however relationship facilities may more likely be used to create links between objects (and versions) in the temporal space.

The database concept is based on a log-only principle (e.g. [Nørv00]). This means that a stored object can never be deleted or updated. Any update of data, including metadata, results in a new object, a new *version* of the old object. A DTD object is stored in many versions as well, each single version reflecting its time's (medical) development, practice, needs, and functionality. This result in a set of evolving DTDs represented in an ascending chronological order.

To never delete objects gives a unique possibility to re-construct any past times of selected periods / intervals. Reading / retrieving objects in a temporal perspective have meaning only

when using a *read date*. If following a linear time axis, as illustrated in figure 2, the read date can be seen as an accurate time line on this axis. The reader may then look at "the world" as it were seen and expressed at that given historical date. The basic philosophy is that any object must be seen in relation to its object date plus the read date.

This approach represents nearly unlimited ways for applications to retrieve (query) and view historical data from the database. Some examples are (cases illustrated in figure 3):

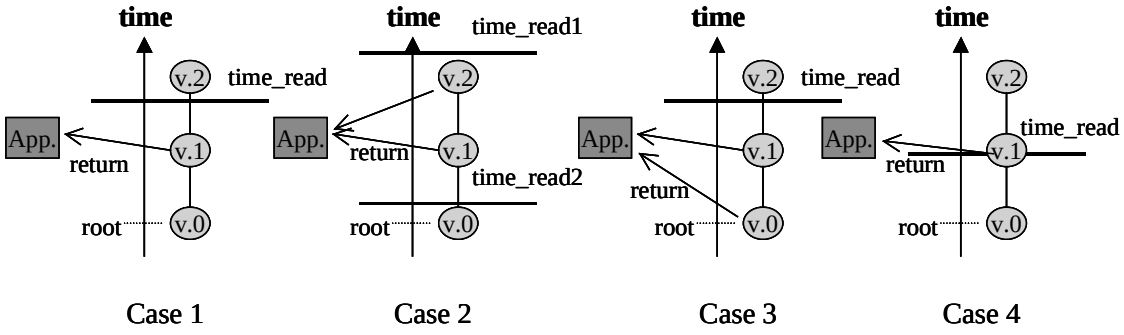


Figure 3: Cases of temporal retrieval of object versions.

- Case 1: Return the must-updated version at the read time - find the version with the newest object date at that time.
- Case 2: Return all versions, or related objects, in the time interval between two read dates, time_read1 and time_read2. All objects older than time_read1 but newer than time_read2 are returned.
- Case 3: Return all the known versions, or related objects, of an object at read time - to go back in time from the read date until the oldest ("root") version is reached. This is a special version of case 2 above.
- Case 4: Return the version that has object date equal to read date if any.

When doing historical search in the database, not only data objects, but also their corresponding DTD versions are available. This helps to give the reader the correct interpretation of the data represented in an old fashion ways.

Relationships or links between objects are allowed. Two types of links exist. First, there are links between versions, or generations, of the same object. Second, an object may have pointers to numerous other objects of any kind. From a database data model view, the first type can be seen as a one-to-one relationship between two generations of an object. The second type may be seen as ordinary relationships of any cardinality between objects of different types (DTDs). These objects may or may not (like weak data types) be independent of each other. However they share the temporal linear viewpoint of the model, which means that they are temporally related to each other in the time.

Going back to figure 2, lets look at the data object version called ‘Data v.1’. This version was generated in the temporal space between ‘Data v.0’ and ‘Data v.2’. ‘Data v.0’ is the *predecessor* of ‘Data v.1’ while ‘Data v.2’ is the *successor* of ‘Data v.1’. Each version,

except root versions, may work as both predecessor and successor. In addition, ‘Data v.1’ has a relationship to a ‘Data add’ object. ‘Data add’ can represent a note or follow-up to the content of ‘Data v.1’. Like in computer-based patient records (CPR), for example, where physicians add their own notes or comments to an earlier stored patient record object.

The object’s object date and optional time stamps show how the objects are related in the temporal space. The implementation of the data object structure is not discussed in the paper.

All together the stored objects including DTDs and their relationships represents a huge and powerful knowledge base.

2.3 DTD objects and versions

Examples of DTDs in a total health database are several. Registered medical standards, nomenclature and terminology may get their own hierarchy of DTD objects. Specific diagnosis with "best choice" treatments and drug prescriptions may be linked to this hierarchy of DTDs. All kind of medical devices may be connected to their own DTD objects. How data are collected and handled, and how measurements are interpreted may be represented in DTDs. Each citizen has their own personal profile object declared by the profile DTDs. Profile DTDs are also created for different actors like physicians of different kinds/branches of medical specialization and all kind of member organizations.

By introducing evolving DTDs, new requirements and functionality can be represented by creating new DTD versions. A new DTD version will replace one or more older versions of the same DTD or extract a set of different older DTDs.

An important aspect is the mapping between DTD versions. The rest of the paper shows examples where such mapping solutions are required between the data representations specified in the set of DTD versions.

3. EVOLVING DTDs

3.1 Interact with the database through evolving DTDs

The following two subsections show how evolving DTDs and mapping between DTD versions can be used for the basic database operations; insertion of new data objects (versions) and reading / retrieving of selected data elements from any stored data objects. In general, the database has to offer smooth functionality for the users, the (user-) applications or programs working on an application level above the database level. A given application is implemented to interact with a given set of DTD versions. In a very long-term temporal perspective, like 100 to 200 years, new application implementations (both software and hardware) will show up rapidly, while the stored (historical) data shall stay stable and accessible. Traditionally, changes in the interface between database- and application level are handled by introducing middleware software, typical implemented to do mapping between specific types of applications and the database. In a large information system there will typically be a set of middleware that in worst case needs to be adjusted every time implementation updates are done either on the database level or on the application level.

Evolving DTDs and the DTD version-mapping functionality may simplify or replace the needs for traditional middleware. An adoption is that over time it is impossible to ensure

that all applications are up-to-date when it comes to use of the newest versions of DTDs. Applications are implemented to interact with specific DTDs, and may keep on using them even if newer DTD versions exist. The other way around, new applications may want to read (historical) elements that are represented by older versions of the DTDs than the new applications are supposed to use.

Subsection 3.2 discusses insertion (write) of new data objects in a temporal perspective by using different time stamps. The temporal framework used is inspired by terms used in [GoOz98]. The following described the use:

- A temporal history allows objects to be associated with time. From this understanding a *temporal historical time stamp* is defined as an (absolute) *anchor* (as temporal structure in [GoOz98], [Sno92]) on a linear time axis, the temporal order. The temporal order is linked to clock / date time of GMT
- A *present time stamp* is defined as a special case of the temporal historical time stamp set to the accurate time / date of today (the time in this moment).

Subsection 3.3 discusses reading of stored data from the database. As mention earlier, a read date is used for a temporal perspective. The read date allows reads from an interval on the linear time axle. From [GoOz98]:

- An *interval* is the duration of time between two specific anchor points (instants) witch are the lower and upper bounds of the interval.

3.2 Insertion of new data objects (write)

The following points are to be concerned when inserting a new object into the database:

- The object date (the creation/transaction time)
- The valid time
- The version/generation of the object
- Relationships to other objects
- Optional extra temporal historical time stamp

How the mechanisms of the object date, versions of objects and relationships between objects are suggested to work is mentioned earlier in the paper.

For any insertion of a data object version the object date is set equal to the present time stamp at the creation (transaction) point. If the valid time differs from the creation time an extra time stamp is included as mentioned earlier. The new data object is then linked to a DTD object. The DTD object validates and stipulates how to interpret the content of the data object. By linking the data object to a given DTD version, this tells what time in the temporal perspective the new object is related to. Logically, a data object can only be linked to a DTD version that exist and is (well) known by the time of the specific object date. If the new object is supposed to work in the time of today, it must be linked to the newest (most-updated) relevant DTD version at the object date. Like if the new object represented the most-updated values of the content (most likely to do), these must work together with the present DTD version. This is because there may exist users/applications searching for

the updated content that anticipate finding the values represented in the way prescribed by the latest DTD. A mapping between DTD versions is required if the application that does the insertion of the new object is referring to an older DTD version. This mapping is shortly discussed in subsection 3.3.

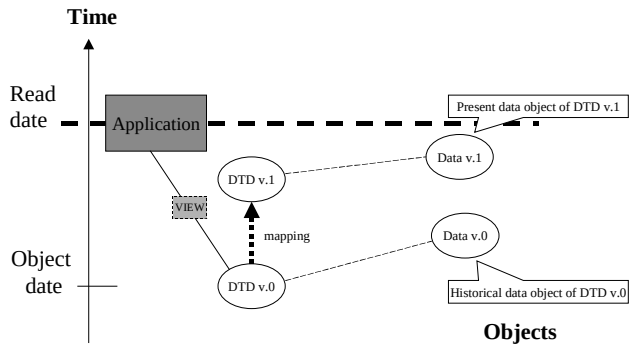
The actions above are sufficient if the new data object is a root version. Like a new blood test that doesn't have any clear references backward on the time axis, but it's of course automatically temporal in the perspective of the owner's lifetime medical history.

If creating a new version of an existing object, a mechanism of versioning must exist. This means to implement a mapping between the new and the old version in the temporal space.

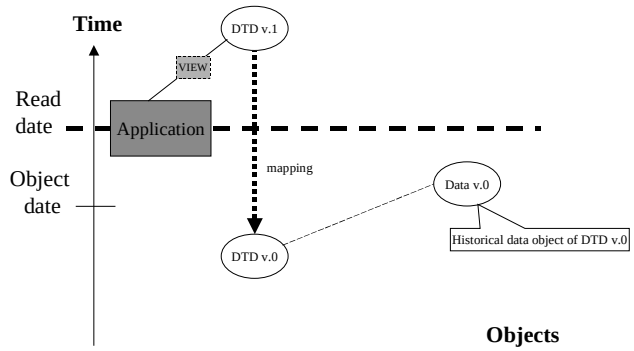
3.3 Accessing the database (read)

A read operation from the database is always temporal by searching an interval on the linear time axis. The upper bound of the interval is set by the read date, which may be seen as a temporal historical time stamp. Setting the read date to present time of today is a special case without being in conflict with the general thoughts below. However, it's not clear how the lower bound is to be set. Several alternatives may be considered. For example three alternative intervals are:

1. The lower bound is set to be zero.
2. The lower bound is set to a given data objects version's object date.
3. The lower bound is set to a given DTD version's object date.



Situation 1: Present time read.



Situation 2: Temporal historical read.

Figure 4: Accessing data from an application level.

A special case in alternative 2 and 3 is to set the lower bound equal the object date of the root version, version 0.

Example of use for alternative 1, also seen in case 3 in figure 3, is generating historical statistics searching all available objects back in time. In alternative 2 the version of the lower bound may include relationships to objects in the temporal interval between the lower- and the upper bound (the read date). For example if the object represents a blood test having relationships to other objects representing laboratory analyses and notes from physicians. Reading the interval

gives data about the whole course of events concerning the test during the interval. In alternative 3 searching the interval between two DTD versions give all registered cases, like diagnoses of a particular disease, of the lower bound DTD version. This might be used in medical research and statistics. Figure 3 shows some other examples as well where the lower bound depends on the returned versions object dates.

It's suggested that only few applications will be able to read an arbitrarily numbers of DTD versions in the temporal time span. Typically, local applications are implemented to access data objects through specific fixed DTD versions, most often the current version. These raise a problem; how shall the database be able to generate temporal results to applications interacting a particular DTD version. A mapping between DTD versions is required. Figure 4 illustrates two situations where an application wants to read data through an "XML view". The "XML view" allows an application to interact with the database thought a given DTD version. In situation 1 the application likes to read the present, most-updated data elements that is older than the read date. These data is found in 'Data v.1' in the figure. If the application is implemented to use the root DTD version, called 'DTD v.0', a 'forward' mapping from 'DTD v.0' to 'DTD v.1' must be able. That is, the database or the "XML view" must prepare a result that the application is able to understand. To do so each requested element from 'DTD v.1' that equals to the similar element in 'DTD v.0' must be found and extracted from the content of the data object of 'DTD v.1'. Any needed data type mappings between the two versions must be included in this process.

In situation 2 the case is the opposite from a temporal perspective. The application is implemented to use 'DTD v.1' but like to read temporal historical data 'backward' to 'Data v.0'.

A simple example for use of situation 1 is if 'Data v.1' represents a personal profile of the owner including an element (or attribute) of his/hers present permanent postal address. If the representation of the postal address has evolved from 'DTD v.0' to 'DTD v.1', like if the zip code element has evolved from 4 to 5 digits, the "XML view" must make sure to still present the postal address the way 'DTD v.0' requires. The example is working for situation 2 as well, just that the mapping is the other way around. If an application wants to figure out the history of where a person has been living during her life so far - it has to search old versions of the data object storing the personal profile. Not all of these old versions may be following the validation of 'DTD v.1', meaning 'backward' mapping must be done in the preparing of a result that the application understands.

Implementation issues for both the "XML view" and the DTD version mapping are due to future research.

4. CONCLUSION AND FUTURE WORK

A future total health information system needs to store and represent data and metadata in a long-term perspective. The paper suggests using evolving DTDs versions in a temporal object model for this purpose. The advantages of storing integrated health data in such a model are several. Integration makes it easier for citizens to control and read their own complete health data history. The temporal chronological order of 'actions' concerning the

owner has many possible usages, like for medical treatments, for legal purposes and for many research and statistically areas.

By representing the data in XML verified by DTDs the stored information is easy to read / retrieve and interpret in the future and easy to exchange between future heterogeneous information systems. However, both XML and in particular DTD is here only used as examples of possible standards. In a very long-term perspective, like 100 years, old standards will continue to be replaced by new ones. We feel that our general temporal model including thoughts about standardized time stamps and mapping between evolving versions easily may fit to future standards as well. Replacing DTDs with XML Schemas will not be a problem. We believe that not every local application manages to be up to date according to new standards for data representation. All applications will not be re-implemented at the same time, and therefore not synchronous on how to represent and transfer data.. If an application is implemented to use specifics standards, like given DTDs, the database can include views that return a readable result generated by mapping between generations of objects and their representation types.

In the proposed temporal data model a linear time axis of the GMT clock / date connecting any object instance to a set of time stamps. Some ideas about how applications may interact with the database through "XML views" are introduced for the functions of insertion and retrievals (read) of stored data and metadata. It's a need for a mapping between given DTD versions since applications are implemented to use fixed specific DTD versions. These mappings must be available in both "directions" on the linear time axis, that means both forward and backward in the temporal space. The general framework may be used for future research on how to represent future integrated EHR in a standardized way suitable for long-term storage and usage.

There are several issues for future work. Insertion of new DTD versions is an important question. The DRL must be able to handle a never-ending stream of new standardized DTDs. Like DTDs for medical usage linked to medical standards, terminology and nomenclatures, or DTDs for more general matters in a future EHR. A formal definition of the mapping between DTD versions is needed and a suggestion for the implementation. Besides the data definition evolution between objects versions there will be a schema evolution as well, from DTD to XML Schema or any other future schema. Any mapping between schemas is not discussed in this paper. Another implementation issue is the "XML view" for interacting between applications and the database.

So far we have only been concerned about the time aspect of the temporal view. In addition it may be interesting to look at how the locality, the geographically position, aspect of each object content may influence the total view and usage of the system.

Finally, traditional database topics like how to query the data and its metadata in an effective way and how data physically are stored may be concerned.

5. REFERENCES

[Bakk00] Bakkeli W, *Stort utviklingspress på journalsystemene: Fra dokumentasjon til kommunikasjon*, news article from http://www.transmit.no/Tmred/emedisin/journal1.htm (in Norwegian).

[BeMu97] van Bemmel JH, Musen MA (ed.). *Handbook of Medical Informatics*, Springer 1997, ISBN 3-540-63351-0.

[BiBl94] Biskup J, Bleumer G. *Reflections on Security of Database and Datatransfer Systems in Health Care*, 13th World Computer Congress 94, Volume 2 1994.

[DTD98] Bosak J. et. al. *Guide to the W3C XML Specification ("XMLspec") DTD, Version 2.1*, http://www.w3.org/XML/1998/06/xmlspec-report.htm

[DiSt97] Dick R.S, Steen E.B, Detmer D.E (Eds.). *The Computer-Based Patient Record - An Essential Technology for Health Care*, Revised ed., Institute Of Medicine, National Academy Press 1997.

[Elvira01] Bellika J.G., Hartvigsen G, Loftesnes L.E., Strandenæs T, *Arkitektur og visualisering, delrapport fra Elviraprosjektet Nettbasert pasient-informasjonssystem*, 2001, The Elvira project, http://www.telemed.no (in Norwegian)

[GoOz98] Goralwalla I. A., Ozsu M. Tamer, Szafron D. *An Object-Oriented Framework for Temporal Data Models*, in Temporal Databases: Research and Practice, pages 1-35, Springer-Verlag, LNCS1399, 1998,

[HCIN00] *HC-INTEREST, Health Care record INTEroperability and Record Structure - Call for Proposals under the Nordunet2 programme*, Reference number 00/1754, Virtual Centre for Health Informatics, Denmark (http://www.v-chi.dk).

[KITH00] *Elektronisk pasientjournal (EPJ) standardisering: Forslag til videre standardisering mv.*, KITH Rapport 13/1, 2000, Norway (http://www.kith.no) (in Norwegian).

[Nørv00] Nørvåg, K. *VAGABOND The Design and Analysis of a Temporal Object Database Management System*, Dr. ing. thesis, Norwegian University of Science and Technology (NTNU), ISBN 82-7984-097-4

[PaGo01] Paton N, Goble C (University of Manchester). *Information Management for Genome Level Bioinformatics*, Tutorials of VLDB 2001, Roma, Italiy.

[ShBa90] Shortliffe EH, Barnett GO. *Medical Data: Their Acquisition, Storage, and Use*, Chapter 2 in Medical Informatics - Computer Applications In Health Care, Addision-Westly 1990, ISBN 0-201-06741-2.

[Sno92] Snodgrass R. *Temporal Databases*. In Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, pages 22-64, Springer-Verlag, LNCS 639, 1992

[XML] http://www.w3.org/XML

[XMLS01] World Wide Web Consortium, *XML Schema Part 0: Primer - W3C Proposed Recommendation, 30 March 2001*, http://www.w3.org/TR/xmlschema-0/

[Waeg96] Waegemann C.P. *The five levels of electronic health records*, in M.D. Computing, Vol.13 No.3,1