Uppsala Master's Thesis in
Computer Science 309
2007-03-22
ISSN 1100-1836

Wrapping Topic Maps in an Object-Relational Database System

Qin Zhang

Information Technology
Computer Science Department
Uppsala University
Box 337
S-751 05 Uppsala
Sweden

Supervisor: Silvia Stefanova
Examiner: Tore Risch

Abstract

The purpose of this thesis project is to develop an XTM (XML Topic Maps) wrapper, called *XTMWrapper*, for the functional and object-oriented DBMS Amos II so that XTM files can be accessed through Amos II using its query language, AmosQL. The wrapper can be used as a database loader for XTM files. The wrapper parses XTM files, translates them into an Amos II data representation, and populates the Amos II database. To represent imported Topic Map data in Amos II, a schema for Topic Maps is developed in terms of its functional data model. The schema represents any Topic Map independent of what the Topic Map describes. The wrapper is written in Java and utilizes a publicly available Topic Map engine, TM4J (Topic Map for Java).

Table of Contents

1 Introduction

Internet makes it easy to access a lot of information. Facing the flood of information accumulated for years people find it more and more difficult to avoid being lost among the vast of information and find the information they need.

In 1999, a new meta-data description representation, called Topic Maps [1], was proposed in order to make the search for information easier. It structurally abstracts information and links relevant information together to enable integration and reuse of information. The Topic Map data model is used to describe how data in Topic Maps is represented and used [2]. The main building elements of Topic Maps are: *topic*, *association*, and *occurrence*, which together form the structure of its data model. Using the data model, people can search among topics to find desired external information described by the Topic Map.

There are several syntaxes for Topic Maps where XTM (XML Topic Map) [3] is a popular one. It imposes XML syntax on Topic Maps web documents. Another popular syntax is LTM (Linear Topic Map) [4]. The present project works only with Topic Maps stored as XTM files. Therefore, it will not work with other Topic Map file formats like LTM [4].

The TM4J project [5] provides a Topic Map processing engine TM4J for creating and manipulating XTM documents. This thesis project develops an XTM data loader based on TM4J.

TMAPI [6] is a commonly used application interface for Topic Maps to access and manipulate data in Topic Maps. TM4J provides interfaces conforming to TMAPI 1.0 alpha release [7] for the development of Topic Map based applications.

Amos II (Active Mediators Object System [8] [9]) is a distributed mediator database system. Its core is a light-weight extensible database management system. Using its functional data model and relationally complete object-oriented querying language

(AmosQL), it allows query and integration among heterogeneous data sources. The integration of data is implemented by the mediator-wrapper approach [8]. Wrappers are responsible for accessing external data sources, while mediators combine the views of wrapped data. Wrappers can be developed by using the interfaces between Amos II and programming languages, such as C, Java and LISP [10]. Wrappers have been implemented for, e.g, Internet Search Engine, CAD Systems XML, and RDF [11].

The task of this Thesis project is to develop a wrapper, called *XTMWrapper*, for Topic Maps stored as XTM files. The wrapper can also be used as a database loader for XTM files. By means of *XTMWrapper* the content of XTM files can be loaded into an Amos II database and accessed by AmosQL. The main body of the wrapper is based on a foreign function in Java. It consists of a front-end and a back-end, exploiting the result of the parser and populating the Amos II database accordingly. A translation between the Topic Maps data model and the Amos II data model was made as a generic schema for Topic Maps in terms of the data model of Amos II. Thus in this project Amos II is used as a repository for Topic Maps and it enables general queries over Topic Maps using AmosQL. In addition, some help functions and procedures are designed for easier navigation in the Topic Map information stored in the database.

This report consists of four chapters. The first chapter offers an overview of background knowledge concerning databases, Amos II, and Topic Maps. It is followed by the general architecture of the developed system, which gives a high-level view of how the system works. Then the implementation of the wrapper is explained in details. At the end, the report is summarized with conclusions (from the past) and (a preview to the) some future work ideas.

2 Background

This chapter gives an overview for the related technologies helpful to make this project, including database systems, Topic Maps, and Amos II.

2.1 Databases

“Database” is a popular word today that participates in all life aspects, providing assistance in various services, such as banking, retails, and education.

In a narrow sense, a database (DB) is just a large collection of structured shared data stored on disk for long time, which can be accessed and processed by a set of software tools.

Database Management Systems (DBMS) are data management programs, operating between users and databases. Databases are uniformly managed and controlled by DBMS when they are created, employed, and maintained. Through DBMS, users can easily define, manipulate and recover data, ensure concurrent data access, and provide integrity and security.

DBMS and database applications use database languages to communicate with the database. The most common database language is SQL.

A database schema is a structural description of the objects and their relationships in the database [12]. It is created when a database is defined but can be modified as the database evolves.

There are a variety of Data Models, which are languages for describing database schemas on the logical level. The most common ones is the relational model where all data is represented as tables.

Object-relational databases are a combination of relational databases and object-oriented databases. They allow developers to integrate the databases with customized data types, index structures, and query optimizers.

Fig. 1 illustrated system structure of DBMSs. Applications and users interact with the DBMS using the query language SQL or some other query language.

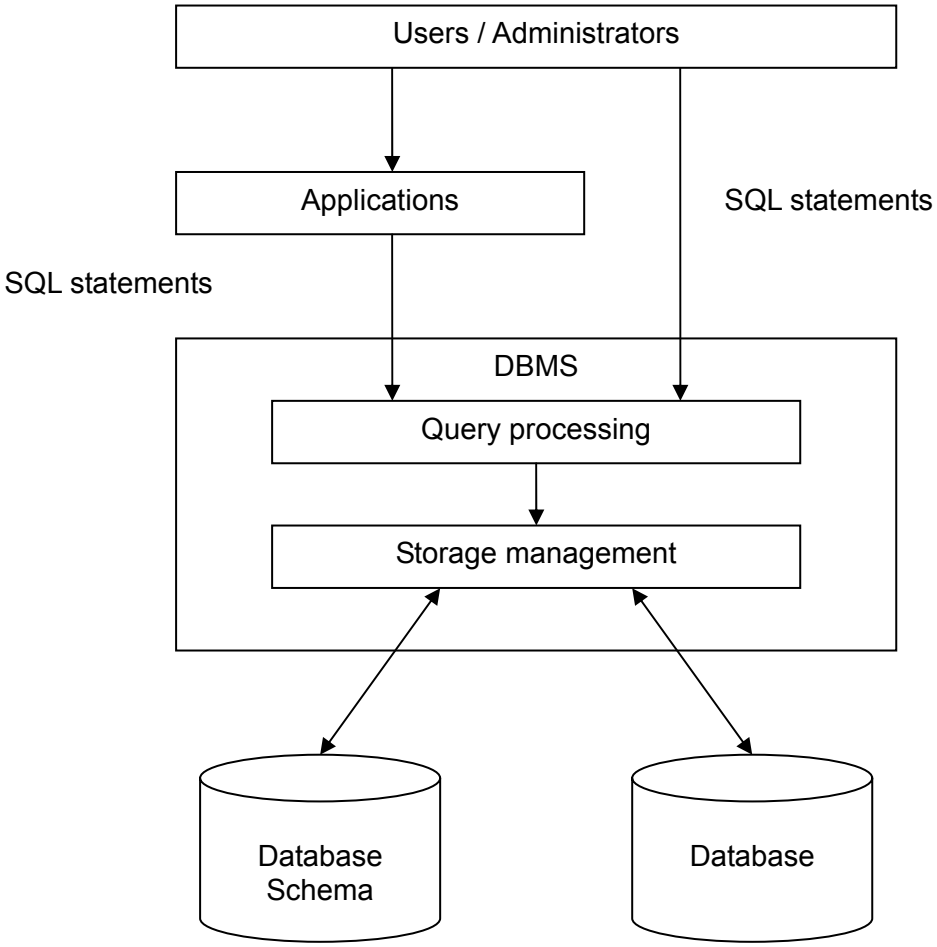

Figure 1: The architecture of a DBMS

2.2 Amos II

Amos II is a distributed mediator database system with a functional data model and a relationally complete functional query language named AmosQL [8]. The core of the system is a light-weight extensible database management system, performing queries and integration on heterogeneous data sources. Amos II employs the mediator-wrapper approach to integrate external data. The wrappers are system extensions to

query and access external data sources, while the mediators combine the views of wrapped data.

2.2.1 Wrappers

Amos II wrappers are pieces of programs knowing how to process queries to external data sources and translate the external data to the local data model that Amos II uses. It interfaces the Amos II kernel and the external data sources, providing a way to transparently access the external data sources. There are already wrappers for Internet Search Engine, CAD Systems, XML, RDF, etc. [11].

A wrapper interfaces a particular kind of external data sources by calling particular API or query languages recognizable by the sources. It should contain the knowledge about the schema and meta-data about the source data, as well as a translation rule mapping from the external data model to the Amos II data model. In this project, the XTM files are accessed through a wrapper implemented by the software TM4J [5] to parse XTM files. The imported data is translated into Topic Map data representation in a generic Topic Map Amos II schema that can store any Topic Map.

Foreign functions (sec. 2.2.5) are external functions implemented in some regular programming language, e.g. C, Java, LISP, etc.. They are the basis of wrapping external data sources from Amos II and provide the low level interfaces to external data sources, hence the main part of wrappers. Java programs call Amos II through the *callin* interface and foreign functions can be defined in Java through the *callout* interface. The present system provides a Java foreign function "*loadXTM*", which is defined through the Amos II *callout* interface for Java and calls Amos II functions to populate the database with XTM data through the *callin* interface.

2.2.2 Data Model

Amos II has a functional data model, whose primary elements are: *objects*, *types* and *functions*.

Objects model all entities in the database, including user-defined objects and Amos II system objects. There are two primary kinds of objects: *surrogates* and *literals*. Surrogate objects are user or system defined objects with corresponding object identifiers (OIDs). Literal objects are build-in objects which are maintained by the system without an explicit OID, such as integers and strings. When defining the generic schema for Topic Maps in Amos II, all the Topic Map items are treated as objects populating the same generic Topic Map schema.

Every object is instances of one or several *types*, including meta-objects representing the types themselves. Types are organized in a hierarchy of super-types and sub-types. This means that every object can belong to a set of types, among which there is one *most specific type* assigned to the object when it's defined. *TopicMap*, *Topic*, *Occurrence* and *Association* are some types in the generic schema for Topic Maps in Amos II.

Functions can be used to model object properties, operations over objects and relationships between objects. A function consists of two parts: the *signature* and the *implementation* [8]. The signature defines the function name along with the types and names of the argument(s) and the result. The implementation indicates how to perform the necessary operation given the argument(s). Having the same name, *overloaded* functions can be defined differently depending on their implementation. Different *resolvents* of overloaded functions are differentiated by their argument types.

Depending on their implementation, functions can be classified into *stored*, *derived* and *foreign* functions [8].

- *Stored function*s represent properties of objects which are stored locally in an Amos II database. Attributes of Topic Map items are modeled as stored functions when translating from Topic Maps into Amos II.
- *Derived function*s are functions defined in terms of queries over other Amos II functions. Some derived functions are defined to help the users search and navigate the information of the loaded XTM file.
- *Foreign function*s are functions implemented in external programming languages such as C, Lisp and Java. The main part of *XTMWrapper* is a foreign function *loadXTM* in Java which loads XTM files into the database.

2.2.3 AmosQL Queries

Queries in AmosQL have the format of *select* statements:

select <result>
from <type extents>
where <condition>

In general the semantics of an AmosQL query is as follows [8]:

1. Form the Cartesian product of the type extents.
2. Restrict the Cartesian product by the condition.
3. For each possible variable binding to tuple elements in the restricted Cartesian product, evaluate the result expressions to form a result tuple.
4. Result tuples containing NIL are not included in the result set.

To avoid the inefficiency execution of queries, it's necessary to perform query optimization first to transform the queries into an efficient execution strategy.

2.2.4 Foreign Functions in Java

In order to create a foreign function in Java, the following three steps should be followed [10]:

1. Define Java code to implement the function.
2. Define the foreign function signature in AmosQL.
3. Specify optional *cost hints* to estimate the cost to executing the function.

2.3 Topic Maps

"Topic Maps is a technology for encoding knowledge and connecting this encoded knowledge to relevant information resources [2]*."*

A Topic Map acts as a meta-data description (i.e. a schema) for one or more information resource(s). It represents the concepts from the resource(s) and connects

them to other relevant concepts inside or outside the resource(s). Therefore, with a Topic Map, people can see a big picture about how concepts are linked to each other, “and focusing on the forest rather than the trees.” [13] This can be illustrated by the following figure 3:

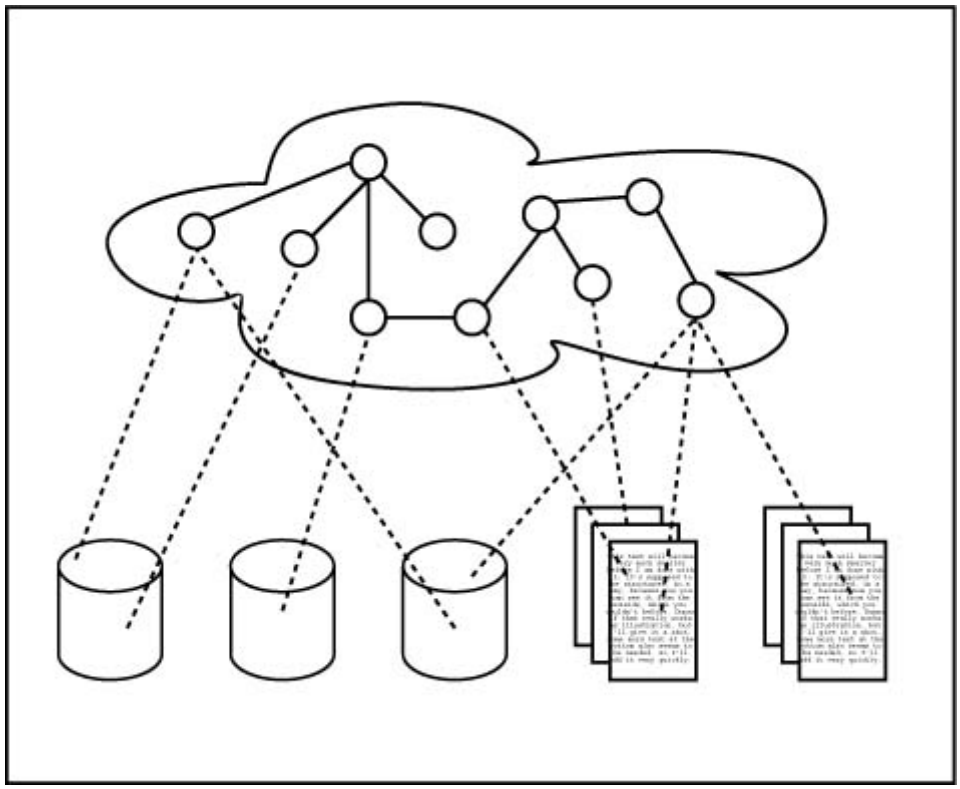

Figure 3: The Topic Map (the cloud at the top) describes meta-data about the information in the documents (the small rectangles) and databases (the small "cans") by linking into them using URIs (the lines) [13]

For interchanging Topic Maps through the internet, XML Topic Maps (XTM) 1.0 [1] is published as a format to represent them using an XML based syntax by TopicMaps.Org. It is revised by ISO to become XTM 2.0 [14]. Since the TM4J is based on XTM 1.0, the *XTMWrapper* system only deals with XTM 1.0 DTD and specification.

The TM4J project [5] provides a Topic Map processing engine TM4J for creating, parsing, and manipulating XTM documents. This project utilizes it for parsing XTM syntax and building a Topic Map representation in Amos II.

2.3.1 Topic Map Data Model

A traditional index entry, e.g. “Le Fabuleux destin d'Amélie Poulain, page 7, See also actress Audrey Tautou”, has three important elements: Topic “Le Fabuleux destin d'Amélie Poulain”, occurrence “page 7” and association “actress Audrey Tautou”. Similarly, Topic Maps borrow the basic features from bibliographic indices: *topics*, *associations* and *occurrences*, to represent knowledge structure that exists in the information sources.

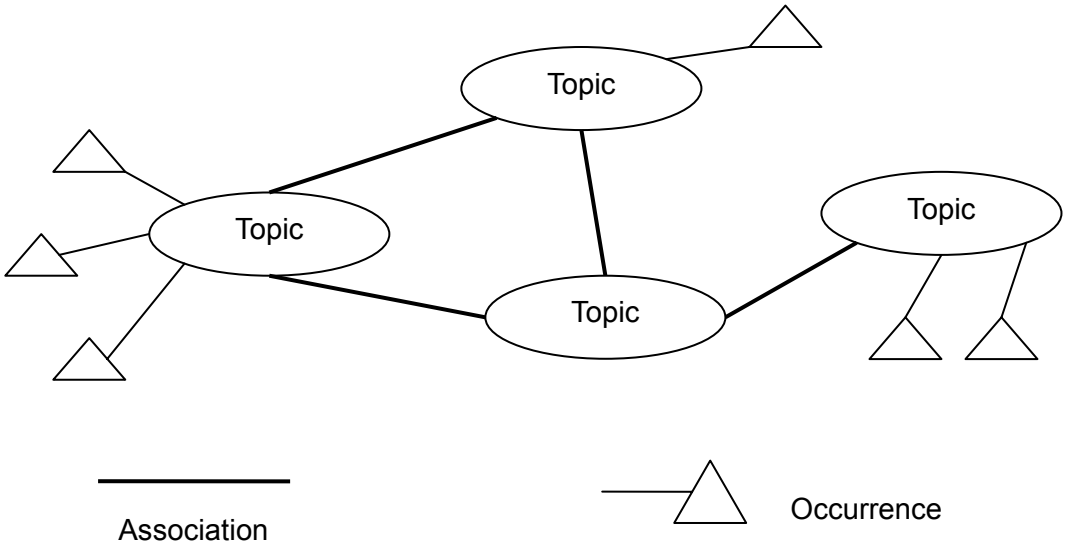

Figure 4: Key Concepts for Topic Maps: *topic*, *association* and *occurrence* [16]

Topics are “symbol(s) used within a Topic Map to represent one, and only one, subject, in order to allow statement to be made about the subject [2].” A topic models the concepts about a subject from the resource(s) on which the Topic Map is applied. In a Topic Map topics are instances of zero or more *topic types*, which correspond to the use of multiple indices in a book for instance [15]. These topic types are also modeled as topics. Topics may or may not have explicit names. For those with explicit names, they have s.c. *base names* as standardized names, and at the same time there can be provided several *variants* of each base name as aliases. Having multiple names for a topic facilitates calling it in different contexts (scopes), such as referring the same object in different languages. Topics also can have *occurrences* and play

roles in associations that will be discussed later.

Sometimes, it's necessary to know which subject a topic models (or *reifies*), especially when merging Topic Maps. Topics about the same subject should be combined so that the subject can be accessed through one topic. The relationship between topics and subjects can be presented by *subject identity*. It is the URI of the subject if it can be addressed in the web; otherwise it should be the reference to its *subject indicator* that is an information resource "to unambiguously identify the subject [2]". Topics sharing the same subject indicator will be merged into a single topic having the combination of properties from all the merged topics.

Occurrences assign relevant information resources to topics. For example, an occurrence can be a picture illustrating the topic, an academic paper studying it, or just some words explaining it. Therefore, the occurrences can be seen as "illustration", "study" and "explanation" of the topics themselves. In XTM, Occurrences are indicated by using URIs. They can also be scoped, which is explained later on.

Associations link together two or more topics, which have some kind of mutual relationships Just like topics and occurrences, an association can also be an instance of zero or more topic types. They are topics themselves. Thus topic types are special cases of association types. Associations make it possible to present all topics having the same relationship, (e.g.) ignoring the specific of each topic and concentrate on the relationship instead. This feature gives a great power to "intuitive" navigation among large data sets [15]. Every topic participating in an association is a member of that association and plays a *role* in it. The role is presented by a topic defining how the member topic acts in that association. In addition, associations contain information about the member topics, so they're multidirectional. This means users need only to know one of the member topics in order to navigate to all other members. Associations can also be assigned different scopes.

It was mentioned above that names of topics, occurrences, and associations need to be assigned in certain context, i.e. their *scope*. People need context to understand things, for example, a word can have different meanings in different context. So does the computer. The scopes offer the contexts and help the computers to process the Topic

Maps without ambiguities. Moreover, scopes can also aid navigation [15].

Topic Maps build up multi-directional navigation paths crossing the immense topic space and covering several knowledge fields. With the Topic Map data model knowledge and information can be integrated, structured and managed; and navigation among the ocean of information will become easier.

3 Architecture

This chapter presents an overview of the developed *XTMWrapper* system. It starts with a high level view of the architecture and then gives examples of how users interact with the system.

3.1 Architecture

The following picture shows the abstract architecture of *XTMWrapper*.

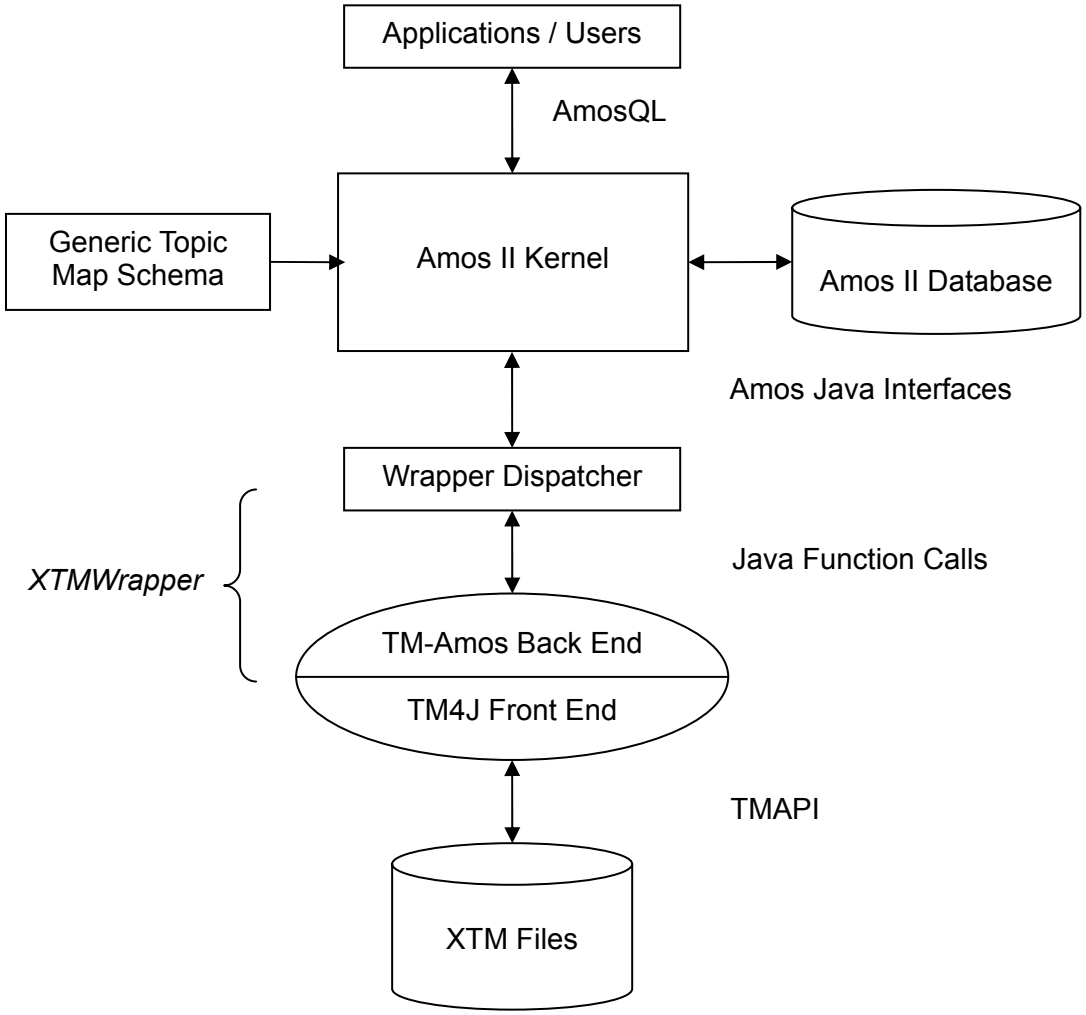

Figure 5: High level architecture of *XTMWrapper*

A short description of the blocks of the architecture is given as follows.

- *Wrapper Dispatcher* receives commands and arguments, by the users, through

Amos II and dispatches the work to the *XTMWrapper*.

- *TM4J Front End* contains two classes *XTMParser* and *XTMBuilder*.
  - *XTMParser* reads the XTM files, parses the syntax and forwards the tokens and data to *XTMBuilder*.
  - *XTMBuilder* checks the XTM syntax, collects all the tokens and builds up a Topic Map main memory data structure in Java.
- *TM-Amos Back End* consists of two classes, which are *TopicMapWalker* and *XTMWrapHandler*.
  - *TopicMapWalker* traverses the Topic Map main memory data representation and calls *XTMWrapHandler* to handle the specific events according to the ongoing tokens.
  - *XTMWrapHandler* deals with the translation between Topic Map objects and Amos II objects, and then populates the Amos II database.

TMWapper and *Amos II* are connected through the *callin* and *callout* interfaces for Java [10]. Amos II calls a foreign function, *loadXTM*, to load the XTM files into the Amos II database through the *callout* interface. This foreign function in its turn calls other Amos II functions to populate the database through the *callin* interface.

Users send requests to Amos II by giving the URLs of XTM files they want to query as the argument to the foreign function *loadXTM*. Then Amos II transparently accesses the desired XTM files, store their information in the database and responses to the users' queries.

3.2 Interaction with the *XTMWrapper*

The main overloaded function to load XTM files into the database is:

loadXTM (Charstring file, Charstring baseURL) ---- loadXTM[1]

loadXTM (Charstring file) ---- loadXTM[2]

loadXTM[1] takes two arguments both of type *Charstring*. The first argument *file* indicates where to find the XTM file the user wants to query. It can either be a local file address or a remote URL on the Internet. The second argument *baseURL* sets the URL property of topics for that Topic Map.

loadXTM[2] is a derived function. After receiving the file address from the users, it calls *loadXTM*[1] and passes it together with an empty string (as the baseURL) to *loadXTM*[1]. In this case, the *baseURL* of the topics will be the same as *file*, i.e. the name of the XTM file.

```
Usage Example:
/*Load an xtm file from local disk given the baseURL.*/
> loadXTM("jill.xtm", "http://martinpc.it.uu.se/jill.xtm");
"Start loading jill.xtm ...."
"XTM Loaded."

/*Load an xtm file from the Internet without specifying the baseURL.*/
> loadXTM("http://www.isotopicmaps.org/tmql/tmql-resources.xtm");
"Start loading http://www.isotopicmaps.org/tmql/tmql-resources.xtm ...."
"XTM Loaded."
```

In order to facilitate easier queries, the following derived functions and database procedures are defined as help functions and procedures. By calling them, users can navigate into the details of the requested XTM file.

- *getTopicID* retrieves the id attributes for all topics or topics belonging to certain Topic Map:
  getTopicIDs ( )->Bag of Charstring
  getTopicIDs (TopicMap)->Bag of Charstring

Usage Example:

> getTopicIDs ( );	> getTopicIDs (:tm1);
"jillstm-topic"	"jillstm-topic"
"short-name"	"short-name"
"developer"	"developer"
"company"	"company"
"description"	"description"
……	……

- *getTopic* retrieves a topic object, given its id attribute:
  getTopic (Charstring)->TM_topic

Usage Example:
> getTopic ("tmql");
[OID 1151]
[OID 1251]
Note: Theoretically, Topic IDs are unique in one Topic Map. But in practice, external references can have the same "id" as topics. So this query returns two result records with the same ID.

- The following functions retrieve the *baseNameString*[1] for the specific topic or the *baseName* object, respectively, by matching its *baseNameString* property:

  getTopicName (TM_topic)->Charstring
  getTopicName (Charstring)->TM_baseName

[1] A *baseName* is the base form of a topic name. It provides a string *baseNameString* to label a topic [3].

Usage Example:
> getTopicName (getTopic (“tmql”));
”TMQL”
”Topic Map Query Language”

getTopicName (“Topic Map Query Language”);
[OID 1152]

- The following functions retrieve the id of a topic referenced by the *instanceOf* property of a topic, an occurrence, or an association:
  getType (TM_topic)->Charstring
  getType (TM_occurrence)->Charstring
  getType(TM_association)->Charstring

Usage Example:
> getType (getTopic (“tmql”));
“query language”

- The function *getScope* retieves the id of the topic referenced by the *scope* property of a topic, an occurrence, or an association:
  getScope(TM_baseName)->Charstring
  getScope(TM_occurrence)->Charstring
  getScope(TM_association)->Charstring

Usage Example:
> getScope (getTopicName (“TMQL”));
“acronym”

- The following functions help users navigate in a Topic Map database. (Usage examples and results will be given later.)
  showTopic(TM_topic)-> Bag of Charstring
  showName(TM_topic)->Charstring

showOccurrence(TM_topic)->Charstring

showAssociationAbout(TM_topic)->Charstring

The following example uses part of an XTM file to show the navigation in a Topic Map. This part of the Topic Map talks about the film “Le Fabuleux destin d'Amelie Poulain” and its actress “Audrey Tautou”.

```
<topic id="amelie">
   <instanceOf>
      <topicRef xlink:href="#film"/>
   </instanceOf>
   <baseName>
       <baseNameString> Le Fabuleux destin d'Amelie Poulain </baseNameString>
   </baseName>
   <occurrence>
      <instanceOf>
         <topicRef xlink:href="#official_site"/>
      </instanceOf>
      <resourceRef xlink:href="http://www.amelie-lefilm.com/"/>
   </occurrence>
</topic>

<topic id="a_tautou">
   <instanceOf>
      <topicRef xlink:href="#person"/>
   </instanceOf>
   <baseName>
      <baseNameString> Audrey Tautou </baseNameString>
   </baseName>
   <occurrence>
      <instanceOf>
         <topicRef xlink:href="#official_site"/>
      </instanceOf>
      <resourceRef xlink:href="http://audrey-tautou.org/"/>
   </occurrence>
</topic>

<association>
   <instanceOf>
      <topicRef xlink:href="#role"/>
   </instanceOf>
   <member>
      <roleSpec>
         <topicRef xlink:href="#actress"/>
      </roleSpec>
      <topicRef xlink:href="#a_tautou"/>
   </member>
   <member>
      <roleSpec>
         <topicRef xlink:href="#film"/>
      </roleSpec>
      <topicRef xlink:href="#amelie"/>
   </member>
</association>
```

Suppose a user interested in this film doesn't know the name of the actress and wants to know more about the actress. She could use *XTMWrapper* to perform the following navigation.

1. Get the topic about the film

Query with help functions and procedures:

```
> loadXTM (”Amelie.xtm”);
> select t into :film
  from TM_topic t
  where getTopicName (t) = “Le Fabuleux destin d'Amelie Poulain”;
> showTopic (:film);
```

Result:

”ID: amelie”
“URL: C:\Amelie.xtm”
“Type: film”
“Name: Le Fabuleux destin d'Amelie Poulain”
“occurrence”
“ http://www.amelie_lefilm.com/”
“ Type: official_site”

Query with “select” and “where”:

```
> select t from TM_topic t;
> select t from TM_topic t where id(instanceOf(t)) = ”film”;
”amelie”
> select t into :film from TM_topic t where id(t) = “amelie”;
```

2. Get the association about the film

Query with help functions and procedures:

```
> showAssociationAbout (:film);

Result:
   ”Association”
   “  Type: direct”
   “  Member: amelie”
   “    Role: film”
   “  Member: j_p_jeunet”
   “    Role: director”
   ”Association”
   “  Type: role”
   “  Member: amelie”
   “    Role: film”
   “  Member: a_tautou”
   “    Role: actress”
```

Query with “select” and “where”:

```
> select a into :role
   from TM_association a
   where id(player(member(a))) = “amelie”
            and id(instanceOf(a)) = “role”;

> select id(t)
  from TM_topic t
  where t = player(member(:role))
            and id(roleSpec(member(:role))) = “actress”;
“a_tautou”
```

3. Get the topic about the actress

Query with help functions and procedures:

> showTopic (getTopic(“a_tautou”));

Query with “select” and “where”:

> showTopic (select t from TM_topic t where id(t) = ”a_tautou”);

Result:

”ID: a_tautou”

“URL: C:\Amelie.xtm”

“Type: person”

“Name: Audrey Tautou”

“occurrence”

“ http://audrey-tautou.org/”

With the help of *loadXTM* and the help functions, users can access and search XTM files through Amos II.

4 Implementation

This chapter describes in details the implementation of the wrapper system in two parts:

- Creating a generic schema in Amos II for the XTM data model.
- Populating the Amos II database with XTM data.

Pictures and examples are presented in order to explain how the schema translation is done and how the software modules collaborate. At the end there is a discussion about the problems regarding the current implementation.

4.1 Functional Topic Map Schema

In order to load XTM files into an Amos II database, a generic Topic Map schema that represents any Topic Map is defined in Amos II. The following pictures depict how the translation from schema for XTM [3] is done. The relationships are specified by the signatures of the functions corresponding to Topic Map primitives.

Notation Explanation:

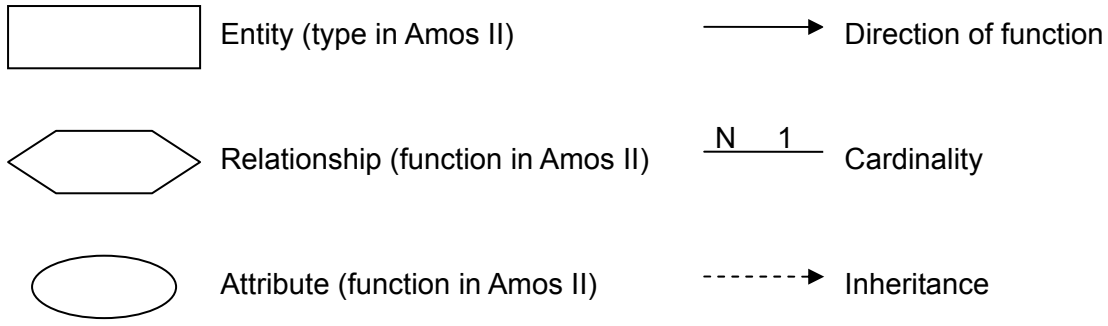

Figure 6: Notation Explantion

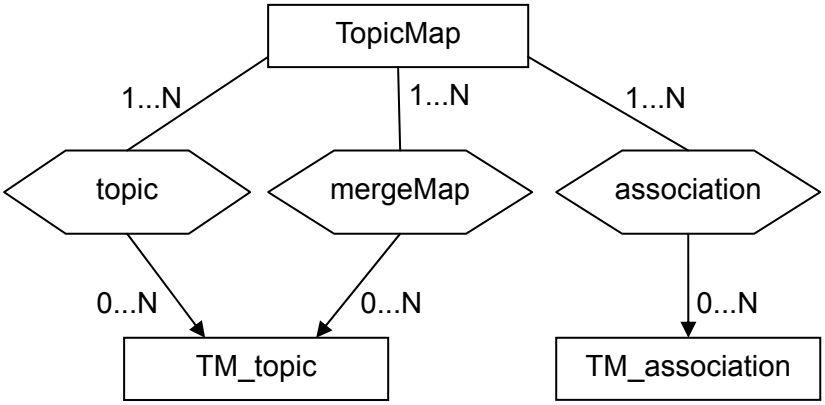

Figure 7: Topic Map schema

Schema:

<! ELEMENT topicMap (topic | association | mergeMap[2]) * >

Function signatures:

topic(TopicMap) -> Bag of TM_topic

mergeMap(TopicMap) -> Bag of TM_topic

association(TopicMap) -> Bag of TM_association

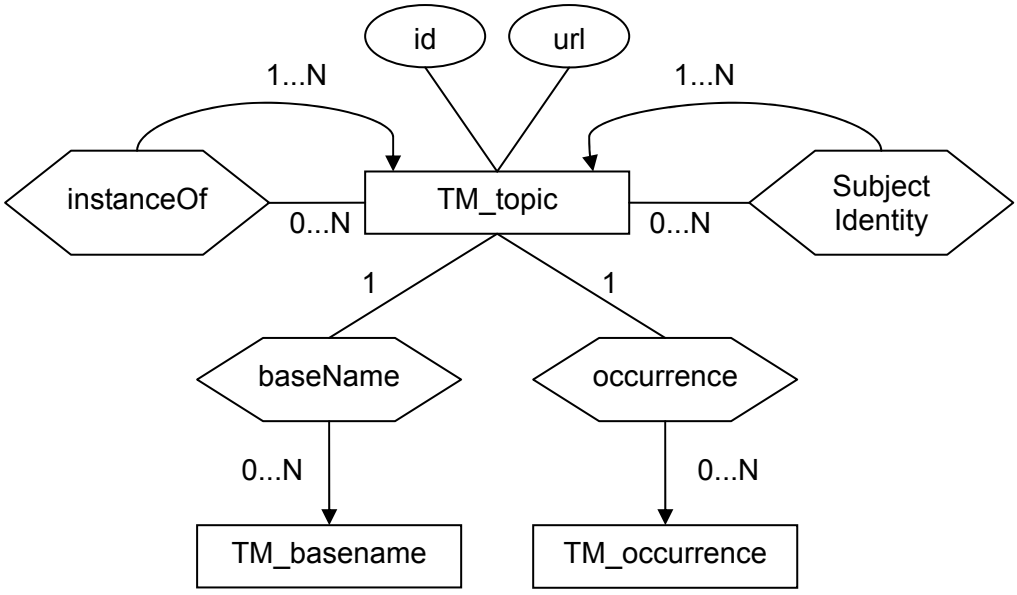

Figure 8: Topic

Schema:

[2] A *mergeMap* references an external Topic Map by a URI. It is a directive to merge the containing Topic Map and the referenced Topic Map [3].

<! ELEMENT topic (instanceOf *, subjectIdentity ?, (baseName | occurrence) *) >

Function signatures:

id(TM_topic) -> Charstring

URL(TM_topic) -> Charstring

instanceOf(TM_topic nonkey) -> TM_topic

subjectIdentity(TM_topic nonkey) -> TM_topic

baseName(TM_topic) -> Bag of TM_baseName

occurrence(TM_topic) -> Bag of TM_occurrence

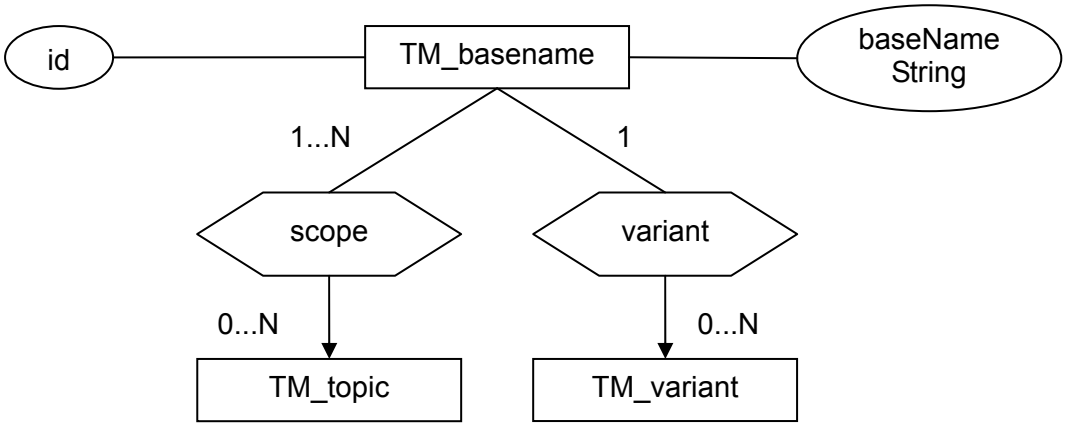

Figure 9: BaseName

Schema:

<! ELEMENT baseName (scope ?, baseNameString, variant *) >

Function signatures:

id(TM_baseName) -> Charstring

baseNameString(TM_baseName) -> Charstring

scope(TM_baseName) -> Bag of TM_topic

variant(TM_baseName) -> Bag of TM_variant key

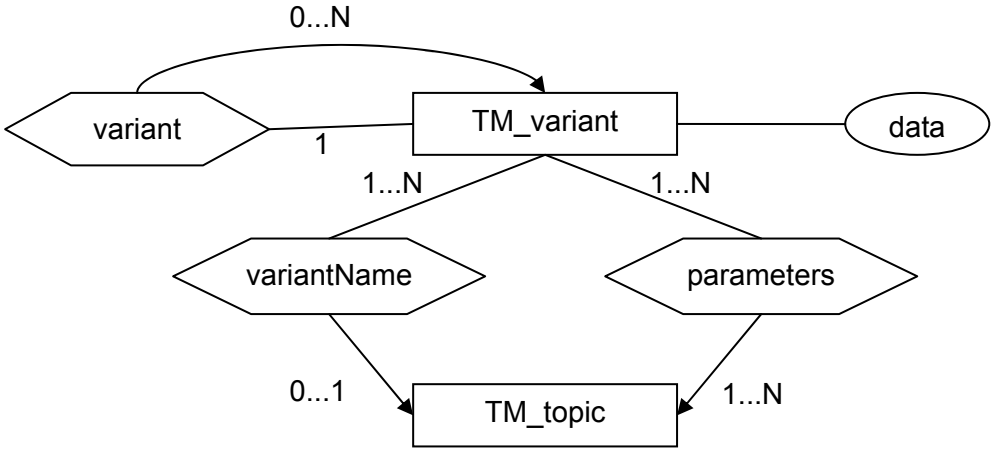

Figure 10: Variant

Schema:

<! ELEMENT variant (parameters, variantName ?, variant *) >

Function signatures:

variant(TM_variant) -> Bag of TM_variant

variantName(TM_variant) -> Bag of TM_topic

parameters(TM_variant) -> Bag of TM_topic

data(TM_variant) -> <Charstring,Charstring>

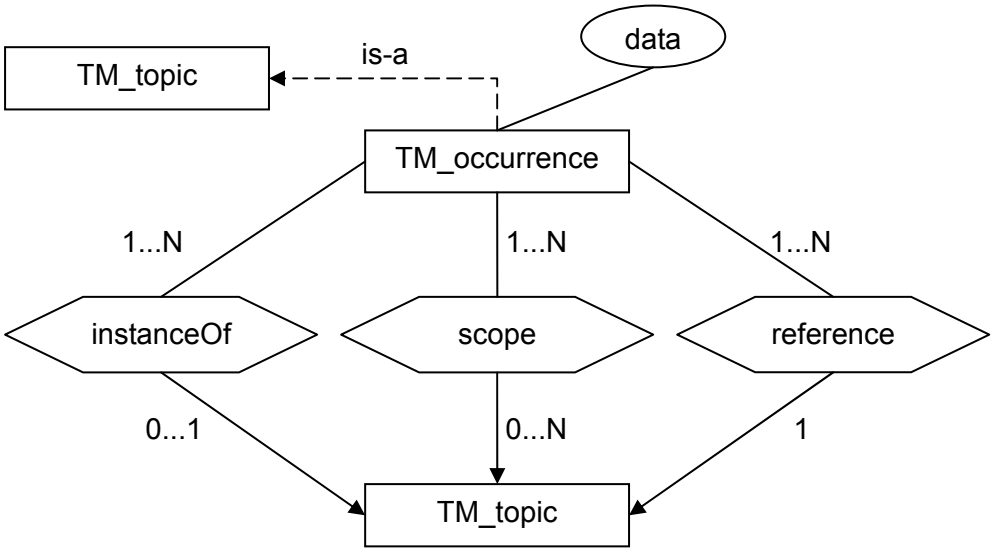

Figure 11: Occurrence

Schema:

<! ELEMENT occurrence (instanceOf ?, scope ?, (resourceRef | resourceData)) >

Function signatures:

instanceOf(TM_occurrence) -> TM_topic

scope(TM_occurrence nonkey) -> TM_topic

data(TM_occurrence) -> Charstring

reference(TM_occurrence) -> TM_topic

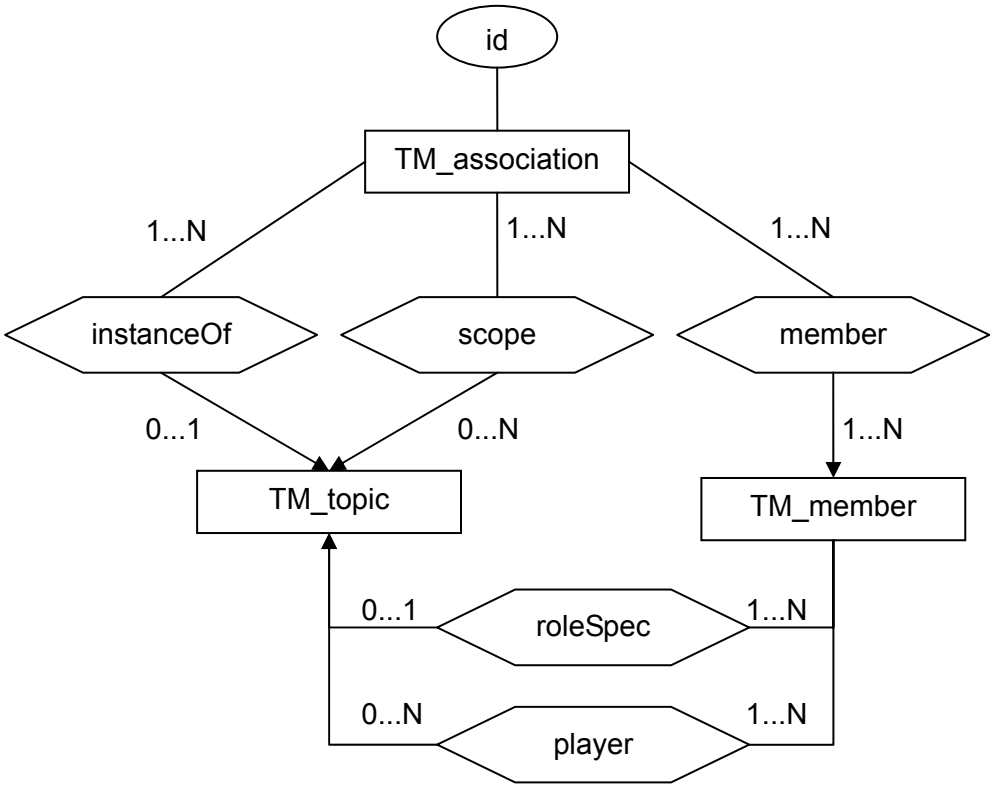

Figure 12: Association & Member

Schema:

<! ELEMENT association (instanceOf ?, scope ?, member +) >

<! ELEMENT member (roleSpec ?, (topicRef | resourceRef | subjectIndicatorRef) *) >

Function signatures:
id(TM_association key) -> Charstring
instanceOf(TM_association) -> TM_topic
scope(TM_association nonkey) -> TM_topic
member(TM_association) -> Bag of TM_member
roleSpec(TM_member key) -> TM_topic
player(TM_member nonkey) -> TM_topic

Notice that this is a generic schema for Topic Maps. It can represent any Topic Map independent of what the Topic Map describes.

4.2 Database Population

The *XTMWrapper* system consists of five primary classes: *Controller, XTMParser, Builder, TopicMapWalker* and *Handler*. The *Controller* class is mostly a work dispatcher and the other four classes undertake the specific work. They communicate with each other and cooperate to populate the database with the data in the requested XTM file. This section will introduce the main functions in each class and how the classes call each other. Figure 13 reveals the respective functionality of the classes and their collaboration as a whole.

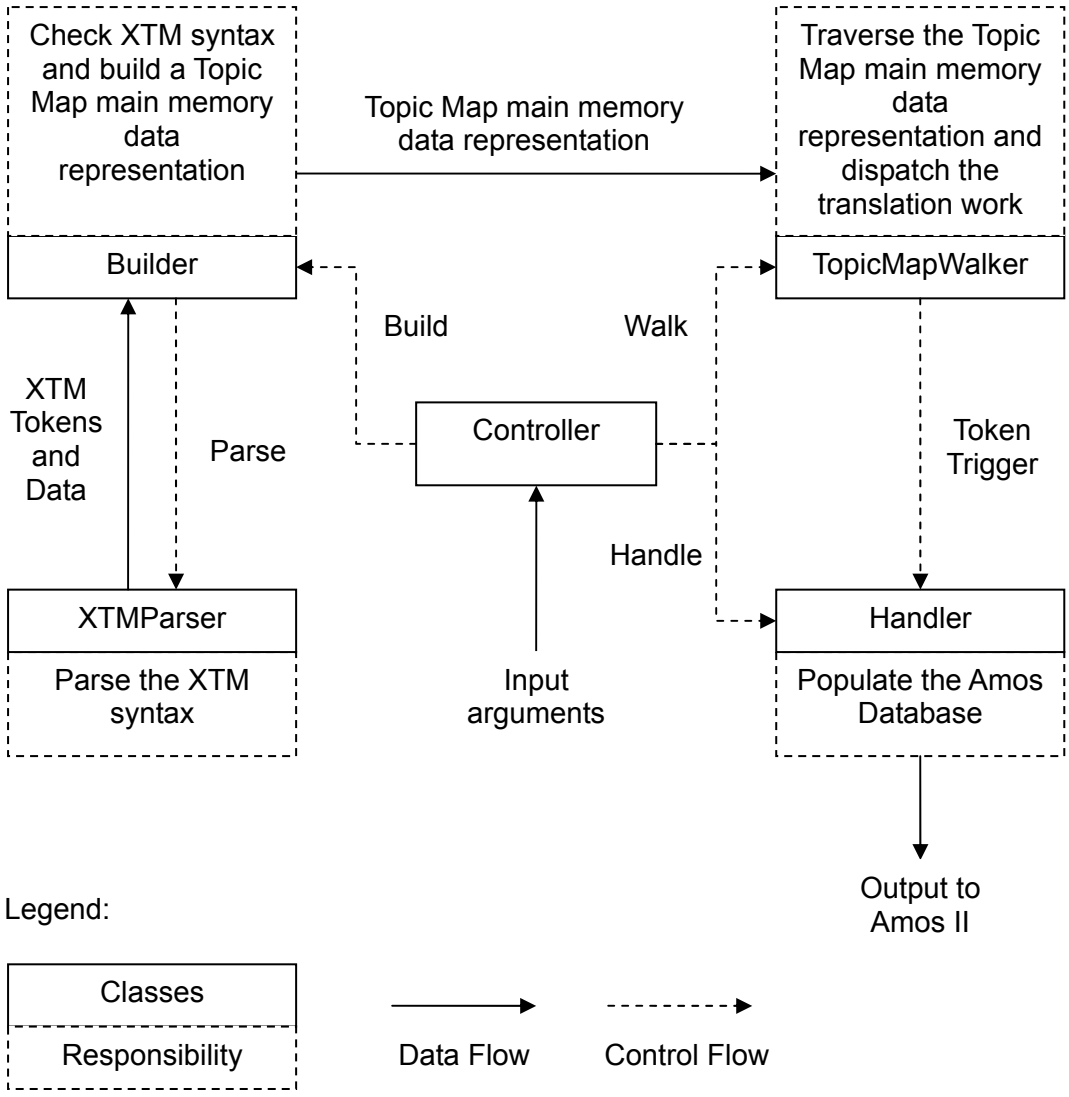

Figure 13: Classes' responsibilities and relationships

4.2.1 Controller class

This class is mainly used as a general controller that receives parameters from the Amos II system, parses them for further use, and calls the corresponding functions from other classes.

Primary methods:

void load(CallContext cxt, Tuple tpl)	
Description	The Java method function “load” implements the Amos II foreign function *loadXTM*. It is as well the main control function in the class. It receives arguments from Amos II and loads the input XTM file. Then it calls other classes to parse the XTM file and populate the database.
Parameter	*cxt* for communicating with the Amos II context [10] *tpl* for holding the argument(s) and the result(s) of the function
Result	If the file is imported successfully, a string indicating the specified XTM file is loaded is returned.

void buildTopicMap (String tmSrc, String baseURL)	
Description	This Java method calls the *Builder* class to build a Topic Map main memory data representation for the file specified by *tmSrc variable*.
Parameter	*tmSrc* for getting the required XTM file *baseURL* for setting the URL attribute of topics.
Result	If no exception happens, a Topic Map main memory data representation will be created and set to the member variable *m_tm*.

void wrapTopicMap ( )	
Description	This method wraps the input XTM file and topics from its *mergeMap* and populates the database.

4.2.2 Builder class

This class is taken from *org.tm4j.topicmap.utils.XTMBuilder* [17] and rewritten. It

parses and imports Topic Map information from XTM files conforming to the XTM 1.0 DTD and specification [1].

Primary method:

void build (InputSource src, Locator srcLoc, TopicMap tm)	
Description	This method parses XTM files and builds the Topic Map main memory data representation for the file specified by *srcLoc*. It's called by the *Controller* class.
Parameter	*src*: Indicates the InputSource to parse and it is passed to the *XTMParser* class. *srcLoc:* Sets the *resourceLocator* of the Topic Map. *tm*: The Topic Map main memory data representation to which parsed Topic Map objects will be added to.
Result	If no exception happens, a Topic Map main memory data representation will be created for the specified XTM file.

Rewritten methods:

String assignID (Locator loc, String id)	
Description	This method generates the id attribute for the Topic Map objects If the id attribute is not specified by the original document, a new id string will be created and assigned to it by *generateID()* method.
Parameter	*loc*: The URL address to be associated with the ID attribute. *id*: The id attribute read from the XTM file.
Result	If *id* is not an empty string, or one object with the same id has not yet been parsed before, an *id string* appended by "ID (as a flag) is returned. Otherwise, a new string is generated and returned.

String assignID(String elementID)	
Description	This method generates the id attribute for the Topic Map object. If the id attribute is not specified by the original document, a new id string will be created and assigned to it by *generateID()* method.
Parameter	*elementID*: The id attribute read from the XTM file.
Result	If an object with this id has not yet been parsed before an id string *elementID* appended by “ID (as a flag) is returned. Otherwise, a new string is generated and returned.

void resourceData (String id, String data)	
Description	This method associates *resourceData* and its *id* to corresponding *variantName* or occurrence. Data and ID are separated by “<”.
Parameter	*id*: The id attribute for this *resourceData* read from the XTM file. *data*: The value of this *resourceData* read from the XTM file.

void ref (String id, int refType, Locator refValue, Locator base)	
Description	This method resolves the references to topics in case of *topicRef*, *subjectIndicatorRef* and *resourceRef*.
Parameter	*id*: The id attribute for the referencing object read from the XTM file. *refType*: The type of the reference: resource, subjectIndicator or topic. *base*: The current valid base locator for the referencing object.

4.2.3 XTMParser class

The class *org.tm4j.topicmap.utils.XTMParser* [18] parses XTM tokens and calls

corresponding functions in the *Builder* class for appropriate processing.

4.2.4 TopicMapWalker class

The class *org.tm4j.topicmap.utils.TopicMapWalker* [19] traverses the Topic Map main memory data representation built by class *Builder* and calls functions in *Handler* for the wrapping. The traverse goes from *topicMap* to topics and then processes associations.

4.2.5 Handler class

This class implements *WalkerHandler* [20] in order to wrap external XTM files and it also populates the database. On the one hand, it is called by *TopicMapWalker* while it goes through the Topic Map main memory data representation. On the other hand, it communicates with the Amos II system and transfers the XTM information. It's the core of the *XTMWrapper* system

Primary methods:

For each element like *topic*, *baseName* and *association*, the class offers one *start* function and one *end* function. Elements are be pushed to a stack when they starts and popped when they end. The stack maintains the order of the elements being processed and provides information for previous elements. The *topicMap* object is always on the bottom of the stack while the current object is on the top of the stack.

Methods:

boolean startTopic (Topic t)	
Description	This method creates a topic object in Amos II and populates its id and url attributes. Then it attaches the topic to the ongoing *topicMap* by setting the function *topic (topicMap)->TM_topic*. Finally, it pushes the topic object into the stack.
Parameter	*t*: The topic to be created.

Result	If no exception happens, the method returns *true*.

void endTopic (Topic t)	
Description	This method pops the topic from the stack.
Parameter	*t*: The current topic.

For other elements like theme (*scope*), type (*instanceOf*) and *roleSpec*, the class offers only one *on* function, i.e there is not "*start*" and "*end*" functions.

Example:

void onType (Topic type)	
Description	This method creates an implicit topic object and populates the *instanceOf (TM_topic / TM_occurrence / TM_association) -> TM_topic* functions in Amos II. The resolvents are decided by the previous element type of the referencing topic.
Parameter	*type*: The referencing topic to be created.

Another method *createTopic* is particularly defined for creating both explicit topics as well as implicit topics. Explicit topics are topics explicitly stated as topic objects; while implicit topics are those referenced from other elements.

Oid createTopic(String id, String address, Locator l)	
Description	This method creates both explicit and implicit topics in Amos II and populates their id and url attributes. It also handles the topics from the *mergeMap*.

Parameter	*id*: id attribute of the topic to be created. *address*: url attribute of the topic to be created. *l*: resource locator attribute of the topic to be created.
Result	If no exception happens, the method returns a proxy object for the topic created in Amos II.

4.2.6 Sequence and Collaboration

Figure 14 is the sequence diagram for the system. It depicts the rough method-calling and message-passing sequence of the *XTMWrapper* system including all five classes presented above. It focuses on the time sequence of the activities of the classes. The *Controller* always makes decision and directs other classes to do what it wants them to do. Firstly, it deals with the input arguments with its member method. Then it calls the *Builder* to build the input Topic Map and the *mergeMaps*. The processed *mergeMaps* are put in a set for wrapping. Thereafter, it creates the *Handler* and connects the *Handler* to *TopicMapWalker* which walks through the input Topic Map and *mergeMaps*. While walking, the *Handler* is called to wrap objects in the Topic Map main memory data representation. That is the most important part of the whole course. There is also a sequence for the wrapping described by the figure.

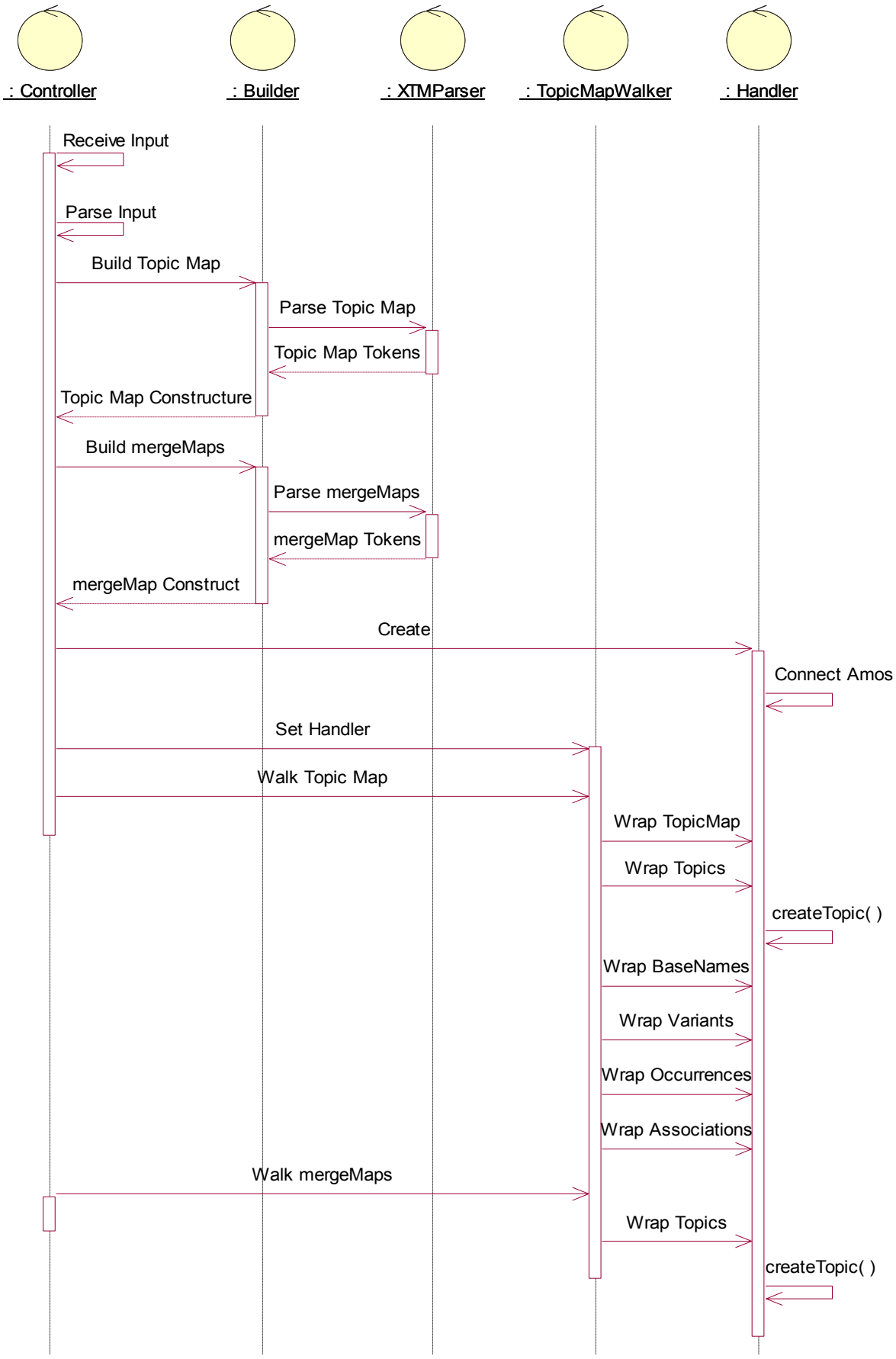

Figure 14: Sequence Diagram for *XTMWrapper*

Figure 15 is the Collaboration Diagram of the *XTMWrapper* system. It focuses on the collaboration of different classes. It also shows the calling and message sequence.

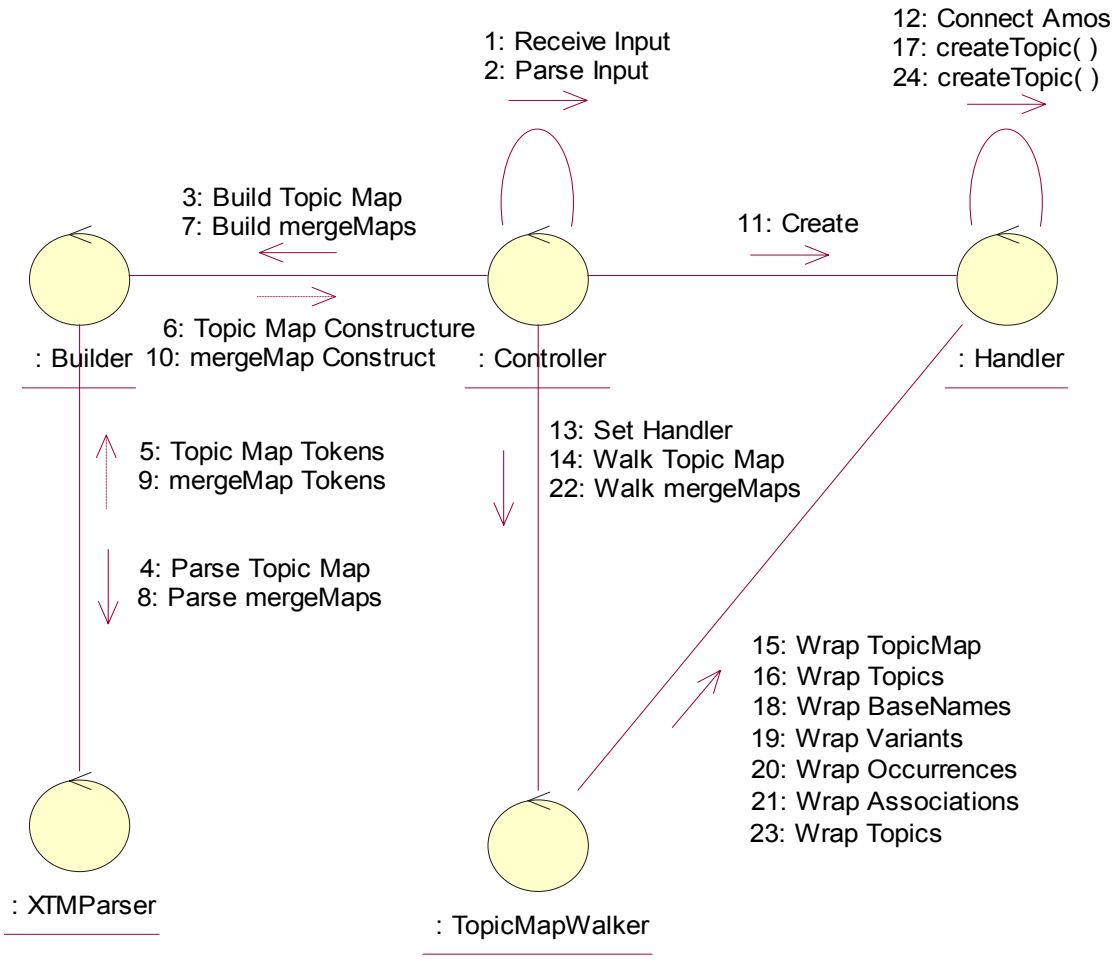

Figure 15: Collaboration Diagram for *XTMWrapper*

4.3 Discussion of Problems

The *XTMWrapper* system accesses XTM files using TM4J. The XTM syntax checking is implemented by those TM4J classes. It works properly most of the time to detect duplicated objects and catch syntax errors. However, sometimes warnings for inappropriate syntax are not detected. For example, there can be only one reference under the tag <*instanceOf*>, according to [3]. Let's suppose we're going to parse the following statement:

```
<topic id="sushi">
        <instanceOf>
                <topicRef xlink:href = "#asian_food"/ >
                <topicRef xlink:href = "#japanese_food"/ >
        </instanceOf>
</topic>
```

The parsing result will be two created <*instanceOf*> intstances and each will have their own reference. The *Builder* simply splits the block of the statement into two. The same happens with the <*baseNameString*>. If there are, for instance, two <*baseNameString*>s (i.e. different strings) under the same <*baseName*> element, the parsing result will be as if there were two separate <*baseName*> elements.

Another example is the case with <*resourceRef*> under <*subjectIdentity*>. According to the XTM DTD [3], there have to be no more than one <*resourceRef*> under a <*subjectIdentity*>. But if it happens to appear more than one <*resourceRef*> tags, the latest one will overwrite the earlier one(s).

The purpose of this project is to make it possible to load XTM files into an Amos II database. A Topic Map main memory data representation is built after parsing. One alternative solution would be, to populate the database while reading tokens from the source file. This would make the performance better, because checking the syntax and building the Topic Map main memory data representation are resource consuming. However, it's very complicated to do syntax checking and correcting and that is why it is simpler with the temporary main memory representation of the XTM file as is done now.

The most important reason for having a temporary main memory TopicMap representation is to handle forward references. That is, often topics created earlier may have to be modified, removed or merged into another topic later when processing the whole file. For instance, two topics having the same *baseNameString* in the same *scope* have to be merged together. (According to the XTM specification, topics in one *topicMap* can not be assigned the same *baseNameString* in the same *scope*.) Suppose

two topics A and B have the same *BaseNameString* in the same *scope*. And A has been created first. Then when B is being processed, the wrapper will remove A from the Topic Map main memory data representation and create a topic as a union of both A and B. The simple populating-while-reading implementation might have problems in such cases. Moreover, such an implementation would have roll-back problem if some syntax error interrupt the program. Therefore, it has been decided to keep the parsing and building implementation as it is in TM4J.

It has been required in the project to include a *baseURL* attribute for each topic. Actually, the Topic Map data model does not define this attribute for topics. In addition, it has to be also taken in account that it's tricky to define a *baseURLs* for topics from *mergeMaps*. The current solution is to set the file address as a *baseURL* for *mergeMap* topics. A possible alternative is to use the *baseURL* given by the user to as a *baseURL* for the *mergeMap*.

The performance of the developed wrapper is not yet tuned. What can be mentioned here is that, it takes some time to build the Topic Map main memory data representation when the XTM file is loaded for the first time. For example, loading the XTM file "http://www.techquila.com/tmsamples/xtm/tmworld.xtm" which contains 562 topics costs 4.6 seconds; while loading "http://www.isotopicmaps.org/tmql/tmql-resources.xtm" containing 108 topics in the same environment costs only 2.0 seconds. It can be further improved by providing a brand new front-end specially customized for Amos *XTMWrapper*.

There are some remaining problems in the current design of the schema for Topic Map Data Model. Let's look at the following example.

```
<association id="employment">
        <scope>
                <topicRef xlink:href = "#1990s"/ >
                <topicRef xlink:href = "#university"/>
        </scope>
</association>
```

The DTD [3], corresponding to such a part of a Topic Map, is:

```
<! ELEMENT association (instanceOf?, scope?, member+ ) >
<! ELEMENT scope ( topicRef | resourceRef | subjectIndicatorRef ) + >
```

It is obvious that there can be no more than one scope under an *association*, while there can be multiple references under one *scope*. Since the schema does not model the scope feature as an object, the solution is to have multiple references under an association with the purpose of scope. So, the constraint on the cardinality of scopes is missing.

The following is another example about how the defined schema works. The XTM DTD [3] requirements for the "*subjectIdentity*" element looks like:

```
<! ELEMENT subjectIdentity ( resourceRef?, ( topicRef | subjectIndicatorRef )* ) >
```

It means there can be only one *resourceRef* under *subjectIdentity* while it is allowed to have multiple *topicRef* and *subjectIndicatorRef*. Since the model doesn't distinguish the purposes of references, the cardinality for topic references under *subjectIdentity* without considering the purpose is still a paradox. The wrapper works currently for *topicRef* and *subjectIndicatorRef* in order to avoid uniqueness violation.

5 Conclusion and future work

The developed *XTMWrapper* enables Amos II to access XML Topic Map files transparently by wrapping XTM data sources to populate the Amos II database. A generic Topic Map Schema was developed that can represent any Topic Map definition. The Topic Map definition is loaded into the database from XTM files using *XTMWrapper*. When XTM files have been loaded into the database their contents can be queried using AmosQL. The development of the wrapper has been based on JDK 1.5 and the Amos II Java interfaces. Some classes from TM4J are used for parsing XTM files and building the intermediate Topic Map main memory Java representation.

The XTM syntax checking and the schema mapping have some limitations and can be generalized, as presented in Section 4.3. Another issue is updating the XTM files loaded into Amos II. One solution is to develop an *XTMWriter* that would write new XTM files. This issue requires further investigations. Wrappers for other Topic Map syntax, e.g. LTM [4], can also be implemented by making a few changes in the schema translation. Finally, the performance can be improved by tuning the program.

6 References

[1]. TopicMaps.Org Authoring Group: XML Topic Maps (XTM) 1.0 Specification

(2001-08-06).
http://www.topicmaps.org/xtm/1.0/

[2]. L. Garshol, G. Moore, JTC1/SC34: Topic Maps — Data Model (2006-06-18).
http://www.isotopicmaps.org/sam/sam-model/

[3]. TopicMaps.Org Authoring Group: XTM 1.0 Document Type Declaration (Normative).
http://www.topicmaps.org/xtm/1.0/index.html#dtd

[4]. L. Garshol: The Linear Topic Map Notation Definition and introduction (v. 1.3).
http://www.ontopia.net/download/ltm.html

[5]. TM4J Project. http://www.tm4j.org/

[6]. TMAPI. http://tmapi.org/

[7]. TMAPI 1.0 Interfaces for TM4J. http://www.tm4j.org/tmapi.html

[8]. T. Risch, V. Josifovski, t. Katchaounov: Functional Data Integration in a Distributed Mediator System in P. Gray, L.Kerschberg, P.King, and A.Poulovassils (eds.): Functional Approach to Data Management - Modeling, Analyzing and Integrating Heterogeneous Data, Springer, ISBN 3-540-00375-4, 2003.
http://user.it.uu.se/~torer/publ/FuncMedPaper.pdf

[9]. S. Flodin, M. Hansson, V. Josifovski, T. Katchaounov, T. Risch, and M. Sköld: Amos II User's Manual.
http://user.it.uu.se/~udbl/amos/doc/amos_users_guide.html

[10]. T. Risch, D. Elin: Amos II Java Interfaces, Uppsala Database Laboratory (2000).
http://user.it.uu.se/~torer/publ/javaapi.pdf

[11]. Uppsala Database Laboratory Amos II Wrappers.
http://user.it.uu.se/%7Eudbl/amos/wrappers.html

[12]. Wikipedia: Database (Retrived on 2007-02-17).
http://en.wikipedia.org/wiki/Database

[13]. L. Garshol: What Are Topic Maps (2002-09-11).
http://www.xml.com/pub/a/2002/09/11/topicmaps.html

[14]. L. Garshol, G. Moore, JTC1/SC34: Topic Maps — XML Syntax (2006-06-19).
http://www.isotopicmaps.org/sam/sam-xtm/

[15]. S. Pepper: The TAO of Topic Maps — Finding the Way in the Age of Infoglut.
http://www.ontopia.net/topicmaps/materials/tao.html

[16]. Wikipedia: Topic Map (Retrieved on 2007-02-17).
http://en.wikipedia.org/wiki/Topic_Map

[17]. TM4J Project: API Documentation — org.tm4j.topicmap.utils.XTMBuilder.
http://tm4j.org/tm4j/docs/apiDocs/org/tm4j/topicmap/utils/XTMBuilder.html

[18]. TM4J Project: API Documentation — org.tm4j.topicmap.utils.XTMParser.
http://tm4j.org/tm4j/docs/apiDocs/org/tm4j/topicmap/utils/XTMParser.html

[19]. TM4J Project: API Documentation — org.tm4j.topicmap.utils.TopicMapWalker.
http://tm4j.org/tm4j/docs/apiDocs/org/tm4j/topicmap/utils/TopicMapWalker.html

[20]. TM4J Project: API Documentation — org.tm4j.topicmap.utils.WalkerHandler.
http://tm4j.org/tm4j/docs/apiDocs/org/tm4j/topicmap/utils/WalkerHandler.html