TMAPI 2.0

Lars Heuer[1] and Johannes Schmidt[2]

[1] Semagia
heuer@semagia.com

[2] INSTANT Communities GmbH
js@sixgroups.com

Abstract. This paper introduces a new generation of the common Topic Maps API (TMAPI) which has evolved from earlier versions based on the Topic Maps Data Model (TMDM) and user experience. TMAPI 2.0 aims to support TMDM and its constraints and to provide a common, user-friendly API for Topic Maps application development independently of a concrete Topic Maps processor.

1 Introduction

Topic Maps API (TMAPI) is a set of Java interfaces and was designed as common programming interface for Topic Maps processors. The initial version was released in the year 2004 and several Open Source and commercial implementations support it. The API was not designed by recognized standards body, but can be seen as a de facto standard for accessing and manipulating topic maps in a portable way. It has been adopted and ported to other programming languages (i.e. PHP5 [5] and .NET [8]) as well.

In the design phase the project members discussed if a programming language neutral approach should be taken for the next TMAPI generation. Even if this idea has its merit it was rejected since each programming language has its own idiomatics and designing an API which meets a common subset of popular languages was felt unpromising. Since the TMAPI project has historically a Java background, the project members opted to focus this language again. Further, the idea that the interfaces should constitute a solid foundation to implement the upcoming standard Topic Maps Query Language (TMQL [3]) on top was also rejected: The project should simply offer an API to access and modify topic maps aligned to TMDM.

Maicher, L.; Garshol, L. M. (eds.): *Subject-centric computing. Fourth International Conference on Topic Maps Research and Applications, TMRA 2008, Leipzig, Germany, October 16-17, 2008, Revised Selected Papers.* (Leipziger Beiträge zur Informatik: XII) - ISBN 978-3-941152-05-2

2 Design Objective

Since the release of TMAPI 1.0 several Topic Maps standards have been published, especially the Topic Maps Data Model (TMDM [2]) must be emphasized here. Because the initial version of TMAPI does not support all facets of TMDM well, the main design objective for 2.0 was TMDM compliance and the observance of its constraints to some extend.

Due to reasons explained in the introduction, TMAPI 2.0 is explicitly Java-centric and requires Java 1.5 since it utilizes generics and variable arguments; translations to other programming languages should be handcrafted to account for respective language specifics. The UML class diagrams for the core and the index package provided by the TMAPI project can serve as starting points for translations to other object-oriented programming languages.

While the first version does not offer any filtering methods (i.e. iterating over the occurrences of a topic by the occurrence's type), the second generation provides simple filters to ease the development of applications. A more advanced filter language was rejected for the time being but may find its way into a subsequent release.

3 Status

The project members have published UML class diagrams which describe the current status of the project. In favour of readability the class methods are omitted.

These UML class diagrams were used as boilerplate for the project's interfaces.

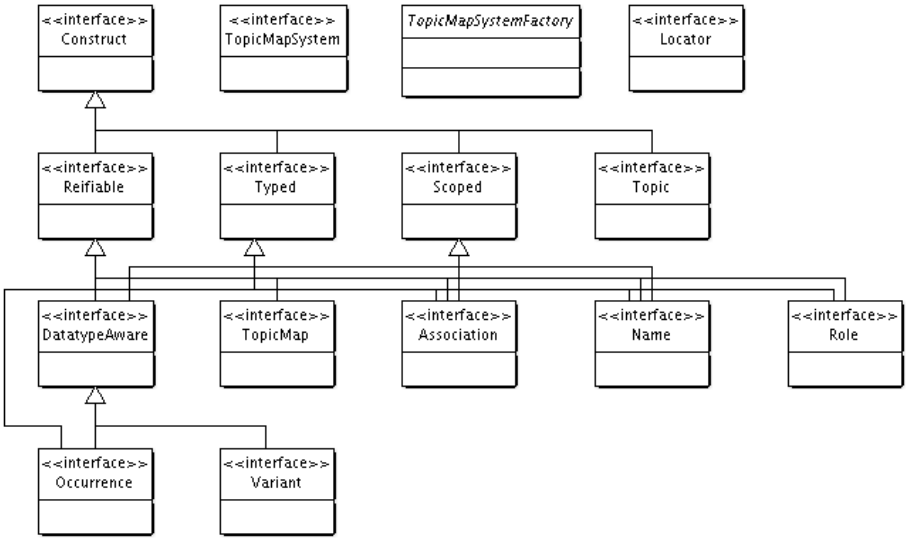

Fig. 1. Abbreviated UML class diagram for the "org.tmapi.core" package

While the first TMAPI version offers just 89 tests to ensure compliance, the new release will provide a suite with approximately 250 tests. The enhanced test suite ensures that different implementations are conform to certain requirements and establishes a profound basis for application programmers to test particular Topic Maps processors against. Further, these tests corroborate the claim that applications which use the project's interfaces are portable over different Topic Maps processors.

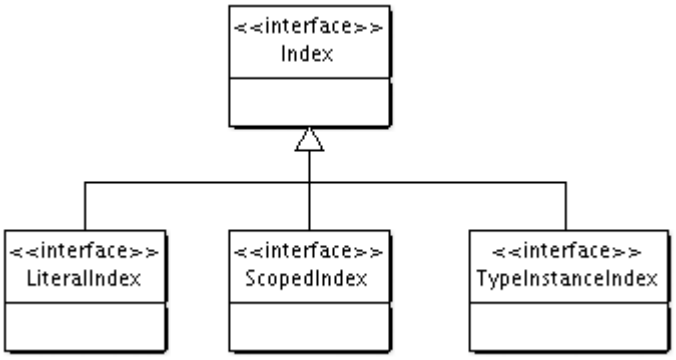

Fig. 2. Abbreviated UML class diagram for the "org.tmapi.index" package

4 Changes

The following sections enumerate important changes between TMAPI 1.0 and 2.0.

4.1 Changes in core

TMAPI 2.0 introduces several generalized interfaces like *Reifiable*, *Typed*, *Scoped*, and *DatatypeAware*. These interfaces avoid redundant method declaration (i.e. *setType()/getType()*, *setValue()/getValue()*, et al.).

Additionally, the *ConfigurableHelperObject* was eliminated since it was only utilized by the *Index* interface. The indices are now available by simply calling *TopicMap.getIndex(Class indexInterface)*.

As mentioned above one objective was to enforce TMDM constraints. Thus TMAPI 2.0 is more restrictive than its predecessor concerning model constraints (i.e. disallows *Role.setPlayer(null)*).

The naming in TMAPI 2.0 is simplified for convenience:

- *TopicName* is called *Name*
- *AssociationRole* is called *Role*
- *Topic Maps construct* is called *Construct* (TMAPI 1.0's equivalent is *TopicMapObject*)

4.2 Changes in index

The main changing covers the reduction to only three indices:

- *TypeInstanceIndex*
- *ScopedIndex*
- *LiteralIndex*

This approach distances from a single construct view to a generalized view on a topic map ("literal view", "typed view", and "scoped view"). From these views specific constructs can be accessed (i.e. return all associations in scope x). Topic Maps constructs are available in multiple indices, i.e. *Occurrence* in *TypeInstanceIndex*, *ScopedIndex*, and *LiteralIndex*. The reduction to three indices makes reindexing and / or synchronization more expensive: I.e. a *TypeInstanceIndex.reindex()* operation has to resynchronize the information about topics, associations, roles, occurrences, and names, while a TMAPI 1.0 *AssociationsIndex.reindex()* would only update the information about associations. However the project members believe that *Index* implementations will rather realize constant synchronization.

The *IndexFlags* interface was abolished. Its only method *isAutoUpdated()* is now available in the *Index* interface.

4.3 Specific changes

DatatypeAware Is the superinterface for *Occurrence* and *Variant*. Therefore it provides several methods for value assignments. It requires the Topic Maps processor to set the datatype implicitly to *xsd:string* in *setValue(String value)* and to *xsd:anyURI* in *setValue(Locator value)*. For convenience, it offers several methods to set and read values where the datatype is implicitly assigned and introduces *setValue(String value, Locator datatype)* in order to be consistent with TMDM's concept of datatypes; *getDatatype()* returns the *Locator* identifying the datatype of the value.

Topic Provides filter methods *getRolesPlayed(Topic type)*, *getNames(Topic type)*, *getOccurrences(Topic type)* which return only those constructs which have the specified type. Further, various factory methods for *Name* and *Occurrence* are provided, inter alia a method for creating names with the default name type.

Association Does not allow `null` for player and type assignments. Further, getRoleTypes() and a method to filter the association roles is provided.

Role Does not allow to set the role player and type to `null`.

TopicMap Provides *getTopicBySubjectIdentifier()* and *getTopicBySubject Locator()* (moved from the index package). Even more importantly, the TopicMap interface does not allow to create topics without any identity, such as an item identifier, a subject identifier, or subject locator.

5 Conclusions and Further Work

The project is currently in alpha status but it should have reached a certain degree of maturity when this paper gets published. TMAPI 2.0 benefits from the meanwhile finalized TMDM. While the previous version supports the XTM 1.0 model [1] and some aspects of TMDM, TMAPI 2.0 has shifted to a TMDM compliant API which also considers programmers' convenience requirements.

Some interesting proposals, like a more advanced filter language or interfaces for TMQL, have been delayed due to lack of human resources and time. Further, TMAPI lacks of a standardized transaction management which seems to be necessary prior TMAPI gets accepted in an enterprise context.

The remaining paper elaborates on the rejected advanced filter mechanism which is meant to bridge a gap between a complete query language and a programming API.

5.1 Filter Language

Even if TMQL is close to be an ISO standard, the success of Mircosoft's LINQ [4] and the recent popularity of domain-specific languages [6] has shown that there is desideratum to have specialized languages available which solve particular problems. Ideally, the developer can stay in the familiar programming language.

The new TMAPI version supports some limited filter methods like navigating from a topic to its occurrences which have a particular type, but these filter methods are not satisfactory for more complex tasks like navigating to all occurrences with a particular type and returning the value if the datatype is *xsd:string*. To accomplish such a navigation, the application developer has to write code against TMAPI which might be tedious or she has to switch to another language like TMQL which requires some learning effort.

A simple, domain-specific filter language should be a good, intermediate solution here: The developer stays in her familiar programming language and uses the usual tools and can utilize type checking performed by the compiler.

Due to lack of resources the filter proposal has not been worked out completely, but the general idea is, that the TMAPI project would provide a new, immutable interface *Filter* which can be passed around to all kind of interfaces which represent a particular Topic Maps construct.

One possibilty to create such a *Filter* would be the mentioned domain-specific language:

```
// Return those role players which play the role "group" in a
// "member-of" association where the current topic plays the
// role "member":

Filter<Topic> filter =
  roles(member).parent(memberof).roles(group).select(player);

for (Topic player: topic.match(filter)) {
    doSomethingWith(player);
}
```

The language used to create the filter should be obvious: The filter takes the current topic as context to navigate to the played roles and compares the role type with the topic "member". For each role the parent association is visited and its type is compared to the topic "member-of". From the association, the filter navigates down to each role of type "group" and selects the player from it.

Even though the domain-specific language leaves room for improvement, the equivalent TMAPI code is certainly longer:

```
// Visit all role the topic plays
for (Role r: topic.getRolesPlayed()) {
    if (!r.getType().equals(member)) {
        continue;
    }
    Association assoc = r.getParent();
    // Compare the association's type
    if (!assoc.getType().equals(memberof)) {
        continue;
    }
    for (Role role: assoc.getRoles()) {
        if (role.getType().equals(group)) {
            doSomethingWith(role.getPlayer());
        }
    }
}
```

Due to the immutability of *Filter* it can be reused in several contexts, while the code on top of TMAPI is not easily reusable unless the developer creates a library for common tasks.

Since not every TMAPI implementation has the necessary resources, the project itself should provide a generic implementation. This default implementation would therefore work with every TMAPI compatible implementation, even if it might not be optimized for the specific Topic Maps processor.

A "service provider interface" would enable TMAPI implementations to provide Topic Maps processor-specific, optimized implementations of the *Filter*.

The authors of this paper regard the filter language with a default implementation as reasonable extension to the current interfaces since it provides rich navigation facilities and reduces development time considerably.

References

1. ISO/IEC. 13250:2003: Information Technology — Document Description and Processing Languages — Topic Maps. Technical report, International Organization for Standardization, Geneva, Switzerland., 2003.
http://www.y12.doe.gov/sgml/sc34/document/0322f iles/iso13250 – 2nd – ed –v2.pdf.

2. ISO/IEC. IS 13250-2:2006: Information Technology — Document Description and Processing Languages — Topic Maps — Data Model. Technical report, International Organization for Standardization, Geneva, Switzerland., 2006.
http://www.isotopicmaps.org/sam/sam-model/2006-06-18/.

3. ISO/IEC. FCD 18048: Information Technology — Document Description and Processing Languages — Topic Maps — Query Language (TMQL) 2008-05-15. Technical report, International Organization for Standardization, Geneva, Switzerland., 2008.
http://www.isotopicmaps.org/tmql/tmql.html.

4. Microsoft Corporation. The LINQ Project, 2005.
http://msdn.microsoft.com/enus/library/aa479865.aspx.

5. J. Schmidt. PHPTMAPI.
http://phptmapi.sourceforge.net/.

6. D. Spinellis. Notable design patterns for domain specific languages. Journal of System and Software, 56(1):91–99, Feb. 2001.

7. TMAPI project. Topic Maps API.
http://www.tmapi.org/.

8. TMAPI4NET project. TMAPI for .NET.
http://code.google.com/p/tmapi4net/.