

Streaming Topic Maps API

Lars Heuer

Semagia
heuer@semagia.com

Abstract. This paper introduces a new, event-based API to create topic maps. It is independent of particular Topic Maps processors and enables developers to convert any resource into a topic map representation with minimal effort.

1 Introduction

The Topic Maps API (TMAPI [12]) is the de facto standard to create and manipulate topic maps in a Topic Maps processor independent way. Even if this API is supported by several Open Source and commercial implementations, it requires some learning effort and is (at least in the upcoming version 2.0) very strict regarding Topic Maps Data Model (TMDM [4]) constraints. The observance of these constraints requires some work if a resource (not necessarily a serialized topic map) should be transformed into a Topic Maps representation. Even worse, this work has to be done for every transformation which leads into repetition of common tasks.

This paper proposes an event-based API which aims to ease transformations of arbitrary resources into a Topic Maps representation with minimal effort. While TMAPI can be seen as Document Object Model (DOM [13]) for Topic Maps, the event-driven API is aligned to the Simple API for XML (SAX [10]) and provides similar advantages over TMAPI like SAX over DOM.

The initial API was implemented in Java but it has been ported to the programming languages Python ([9]) and PHP ([8]). Even if this paper compares the event-API against TMAPI, the proposed API has no relationship to the common Topic Maps API, it serves just as an example to emphasize the difference between a push API and a DOM-alike interface.

2 Design Overview

SAX has proven to be effective and due to its simplicity and elegance it has been ported to several platforms. A parser sends notifications to a handler which processes them accordingly. The parsing process is unidirectional: The parser simply pushes the events to a handler and just forgets about them afterwards. The handler contains the logic to interpret the events and to create something meaningful from it.

The event-based API for Topic Maps does something similar: A parser notifies a handler about information items and their properties. The parser does not take care how these events are processed and ignores merging operations, it simply fires a stream of events. Like for SAX, the pivot for the Streaming Topic Maps API is a handler, called *IMapHandler*. Figure 1 shows an excerpt of callback methods which are used by a parser to send notifications.

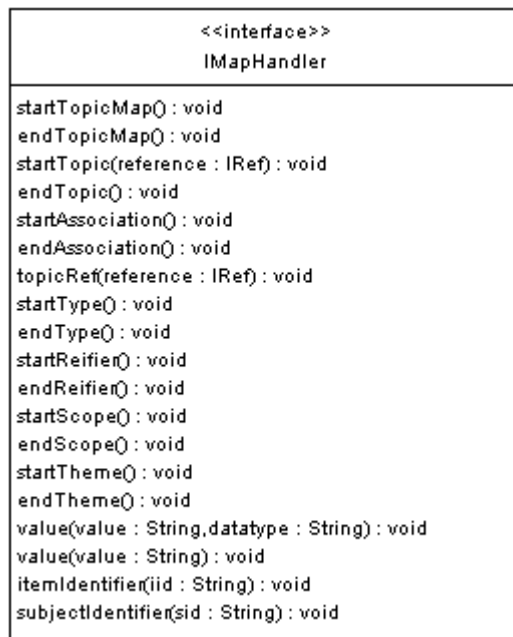


Fig.1. Abbreviated IMapHandler

Altogether, the *IMapHandler* provides 32 callback methods which cover all facets of TMDM. Each information item is introduced with a *start* event and

terminated by an *end* event. The *IRef* interface represents either a subject identifier, a subject locator, or an item identifier and is used to avoid callbacks like *startTopicWithItemIdentifier* or *topicRefWithItemIdentifier*. The *IRef* interface represents always an absolute IRI ([3]). Events like *startType()* and *itemIdentifier(String)* are context sensitive: The IMapHandler implementation has to remember which information item is in the focus and process the notification accordingly.

Given the following CTM [6] fragment:

```
John-Lennon
  - "John Lennon"
.
member-of (member: John-Lennon, group: The-Beatles)
```

A parser reading the fragment will generate a sequence of events like the following (for readability the IRIs are abbreviated to an identifier; the complete IRI for John-Lennon would look like `http://www.example.org/beatles-map.ctm#John-Lennon`):

- *startTopicMap* which is always the first event.
- *startTopic* with an item identifier reference to John-Lennon.
- *startName* to notify the handler that all subsequent events (like *itemIdentifier*) must be interpreted "relative" to the topic name.
- *value* with the string John Lennon
- *endName* to indicate that the name has been parsed
- *endTopic* to indicate the end of the topic John-Lennon
- *startAssociation* to indicate that an association is parsed
- *startType* which indicates that the subsequent topic or topic reference should be interpreted as association type.
- *topicRef* with a item identifier reference member-of
- *endType* precluding the end of the association type processing
- *startRole* Start of a role which participates in the association
- *startType* Role type processing
- *topicRef* with the item identifier reference member
- *endType* indicating the end of role type processing
- *startPlayer* to indicate that the player is processed
- *topicRef* with the item identifier reference John-Lennon
- *endRole*
- ... (processing the other role is omitted)

- *endAssociation*
- *endTopicMap* which is always the last event reported to a handler

As seen, the parser only notifies the handler about events. The parser provides no Topic Maps-related logic, i.e. detecting that a topic with the item identifier `John-Lennon` has already been read and that the latter reference to `John-Lennon` must cause a merge.

It should be emphasized here, that the *IMapHandler* implementation must be done only once for each API (i.e. TMAPI) and that it is relatively easy to implement. To retain the analogy to SAX, the event-based API inverts the responsibility of transforming a resource into a data structure insofar as the parser is variable part here, while the developer has to implement the handler logic in SAX.

The event-driven API allows nesting of events provided that the events cannot be misinterpreted by the *IMapHandler* and that each *start* event has an analogous *end* event. A parser may send a *startTopic* event followed by another *startTopic* event. Due to this flexibility it reduces the development effort for syntaxes which allow nested information items (like CTM). The possibility to interleave events is also the reason why the handler provides callback methods like *startReifier*: After such an event either a complete topic or a reference to a topic may be reported.

As indicated in the introduction, the API is not limited to parse serialized topic maps: A parser can read any imaginable resource and convert it into a Topic Maps representation. An e-mail, an Excel table, the result of a database query, nearly everything is convertible into a sequence of events.

Further, the *IMapHandlers* are stackable: A handler can operate upon another handler and filter the events. If all item identifiers should be omitted, the *itemIdentifier* event is simply consumed by the first handler and not passed on the underlying handler.

3 Push vs. Pull API

Recently, APIs like the Streaming API for XML (StAX [7]) which offer a pulling interface have become popular. These APIs provide typically an iterator over the XML information set and the user is responsible to extract the information which is useful for a particular application. This approach works very well for XML but it seems to be inappropriate for different Topic Maps notations; each Topic Maps syntax provides its own set of features, and creating a common interface for all notations would be a step away from the simplicity of the *IMapHandler*.

Due to the stackability of the *IMapHandler* interface it should be easy to implement filters which omit certain kinds of information items. Based on the fact that topic maps are usually imported as a whole, the pull-approach seems to have no advantage over the API presented in this paper. Pulling information from a resource implies knowledge about the concrete data structure of the resource while pushing the events is much more lightweight and easier to deploy, especially for arbitrary resources.

4 Serializing Topic Maps

SAX can be used to serialize a data structure into a sequence of events which result into XML elements and attributes. Even if the *IMapHandler* can be used like the SAX counterpart, it seems to be more difficult since Topic Maps provides different serialization formats. A syntax like CTM offers nested Topic Maps constructs, while XTM disallows such a structure. How should a Topic Maps processor report the information items?

Due to its outstanding position, the XTM format seems to be the ideal candidate for a pattern in which sequence events are reported, but taking that pattern would lead to a degenerated CTM serialization. The *IMapHandler* for CTM demands other requirements on the order of events as a XTM serializer. Given the fact that there is no common superset on how a topic map should be serialized, the effort to use the *IMapHandler* to serialize Topic Maps has been abandoned.

5 Conclusions and Further Work

The event-driven API has proven to be deployed easily. The pivot *IMapHandler* has to be adapted for the Topic Maps processor-specific API, but it can be used for all kind of parsers.

The Open Source Topic Maps processor tinyTiM [11] uses the event-driven API to import all kind of Topic Maps syntaxes. The author of this paper has implemented a framework, called MIO [2] around the API which allows the discovery of Topic Maps deserializers which use the *IMapHandler*. While porting tinyTiM from TMAPI 1.0 to 2.0, the event-driven API in conjunction with the MIO framework has proven to be simple, since parser logic was not affected: The deserializers use the *IMapHandler* and the TMAPI changes are transparent for them. A deserializer which works for the predecessor also works for the latest version of tinyTiM.

Further, a binary representation of the event sequences, called Binary Topic Maps (BTM [1]), has been implemented. This binary format reduces the storage requirements for serialized topic maps considerable and should be convenient to send the events over a network.

The API has been introduced to the TMAPI project and gained a lot of positive feedback. Unfortunately due to lack of human resources it was not yet integrated into the project. Even though it would be desirable to adopt the event-driven approach widely.

It would be interesting to elaborate the possibility to implement the API on top of a RDF store but this was not yet done. The opposite (a RDF parser which sends events to the handler) is currently implemented.

References

1. L. Heuer. Binary Topic Maps (BTM). <http://www.semagia.com/tr/btm/1.0/>
2. L. Heuer. Topic Maps I/O (MIO). <http://mio.semagia.com/>
3. Internet Standards Track Specification. Internationalized Resource Identifiers (IRIs). <http://www.ietf.org/rfc/rfc3987.txt>
4. 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/>
5. ISO/IEC. IS 13250-3:2006: Information Technology — Document Description and Processing Languages — Topic Maps — XML Syntax. Technical report, International Organization for Standardization, Geneva, Switzerland., 2006. <http://www.isotopicmaps.org/sam/sam-xtm/2006-06-19/>
6. ISO/IEC. FCD 13250-2: Information Technology — Document Description and Processing Languages—Topic Maps—Compact Syntax (CTM) 2008-05-15. Technical report, International Organization for Standardization, Geneva, Switzerland., 2008. <http://www.isotopicmaps.org/ctm/ctm.html>
7. Java Community Process JSR 173. Streaming API for XML (StAX). <http://jcp.org/en/jsr/detail?id=173/>
8. PHP project. PHP. <http://www.php.net/>
9. Python project. Python. <http://www.python.org/>
10. SAX project. Simple API for XML (SAX). <http://www.saxproject.org/>
11. tinyTiM project. tinyTiM. <http://sourceforge.net/projects/tinytim>
12. TMAPI project. Topic Maps API. <http://www.tmapi.org/>
13. W3C. Document Object Model (DOM). <http://www.w3.org/DOM/>