TMRAP – Topic Maps Remote Access Protocol

Lars Marius Garshol

Ontopia AS, Oslo, Norway,
larsga@ontopia.net, http://www.ontopia.net

Abstract. This paper describes TMRAP, an abstract web service interface for remote access to topic maps. It can be used to access a topic map repository to query or update a topic map, or to listen for updates to parts of a topic map. An HTTP binding for the interface is presented in this paper; a SOAP binding will be produced in the future.

1 Introduction

The current stack of Topic Maps standards (data model, interchange syntax, constraint language, and query language) is well suited to building interoperable Topic Maps applications, given that they meet one restriction. That is, applications must be restricted to a single server, because no standardized means for applications to connect to each other over the network currently exists. As has been argued [Moore03], this is unacceptable.

This paper presents TMRAP, a proposed web service interface that aims to make it possible for Topic Maps applications to interoperate over the network, and for systems not based on Topic Maps to connect to Topic Maps servers to add or retrieve data. The version of TMRAP presented in this paper is an extension of the original TMRAP 0.2[Moore04], but is not backwards compatible.

It should be noted that TMRAP is not a standard, but a private proposal, implemented in commercial software. The interface may be freely implemented in other systems.

1.1 Use cases

Designing a web service interface is not like designing an API. Clients using a web service should be able to perform a single task as a single operation, since the overhead of invoking operations is substantial. This is not the case in an API, which has a much higher granularity, since with an API the cost of invoking individual operations is negligible by comparison. This implies that a web service must to a much greater degree make assumptions about the uses to which it will be put, which again implies that a clear picture of the use cases a web service will satisfy is essential to its design.

The use cases which TMRAP was designed to meet are presented in this section. Due to space limitations, the web service is then presented with no explicit arguments as to why it must take the form it does to satisfy these use cases.

Nearly all use of TMRAP involves interchanging fragments of Topic Maps data, which may or may not be processed using Topic Maps-aware tools. If the producer/consumer is not using a Topic Maps-aware tool, using a developer-friendly syntax such as TM/XML may be more suitable [Garshol05]. TMRAP allows the client to choose what syntax to receive Topic Maps fragments in.

Connecting portals A simple, but very powerful example, described in [Pepper04], is that of connecting two portals, A and B, both of which have topic pages about the same topic. When the user navigates to the page for the topic on A, A can send B a request, asking if it knows about this topic. On receipt of a positive answer A can insert a link to the topic page for the same topic in B. TMRAP supports this scenario, and even allows the two portals to share Topic Maps data.

Applications working with fragments A more complex example would be an application that works with only a limited subset (a fragment) of the complete topic map held by the server. An example of this scenario would be when a Topic Maps server is integrated into a portal not based on Topic Maps, running on a different server, which retrieves Topic Maps fragments from the server for presentation when needed. The portal may also submit searches to the Topic Maps server, and present the responses to the user.

Integrating with other applications A very common use of TMRAP would be to use it to integrate a Topic Maps application with one that is not based on Topic Maps, for example a traditional Content Management System (CMS). The CMS maintains metadata about information resources in the CMS. The topic map contains the key metadata from the CMS, together with much additional information (such as what the resources are about (their subjects), and information about the subjects). The CMS would then need to create, update, and delete topics representing the resources as information about these is updated/changed in the CMS [Garshol02b].

This topic map could then be edited in a separate topic map editor to further classify and describe the documents in the CMS. The CMS could also retrieve fragments for these topics and display them (including associations to topics that only exist in the topic map) in a portal rendered from the CMS data.

Creating "knowledge hubs" One very interesting use case is creating "knowledge hubs" where information from many sources is brought together and integrated. This is effectively a form of Enterprise Information Integration (EII) [Halevy05].

In this scenario, information providers invoke operations on a Topic Maps server to add information coming from outside sources, and the server then integrates this information into the existing topic map. Consumers can then query the integrated topic map and retrieve fragments of information extracted from the topic map.

2 The abstract interface

This section defines the abstract interface conceptually, without reference to any specific technology other than Topic Maps. This enables the same design to be used with separate bindings to different technologies, such as HTTP and SOAP. The interface can return fragments representing topics as well as query results from the tolog query language [Garshol05b]; these can be in any Topic Maps syntax, chosen by the client. Authentication and access control are left to the specific bindings.

2.1 Concepts

The interface distinguishes between two actors: a server, containing a topic map repository, and a client. The topic map repository on the server contains any number of individual topic maps, each identified by a unique handle, which is a syntactically opaque string. The server provides access to the repository through a number of operations, each described in a separate section below. No assumptions are made about the client.

Operations are of two kinds: server operations, and callbacks to the client from the server. Each set of operations has a separate section below.

Error handling If a request is in error (incorrect parameters, tolog or fragment syntax error, no topics found, etc) this must be reported to the client. Erroneous requests must not cause any state changes on the server, whether to the topic map or to the list of registered client listeners. The details of error reporting are left to the protocol bindings.

2.2 Server operations

Server operations are invoked by the client, and the interface does not dictate what triggers an operation; this is left to the discretion of the client.

The `get-topic` operation This operation is used to get a fragment representing a single topic. The parameters to the operation are shown in table 1 on the following page.

If the `topicmap` parameter is provided the topic maps identified by it are queried, otherwise the server decides which topic maps are queried. The rationale for this is that if the client knows that it wants to query a specific topic map it can do so. The most common use case, however, is that the client simply knows that it wants information on a specific topic. In this case it can ask a known TMRAP server without having to worry about the internal structure of that server.

There are many possible means by which the server might determine which topic maps to operate on, such as operating on all, on a default topic map, on all currently loaded topic maps, on all topic maps to which the client has access

Parameter	Required?	Repeatable?	Type?	Description
item	no	yes	URI	An item identifier of the sought topic.
subject	no	yes	URI	A subject locator of the sought topic.
identifier	no	yes	URI	A subject identifier of the sought topic.
topicmap	no	yes	String	A topic map handle.
syntax	no	no	String	A string identifying the syntax used in the fragment.
view	no	no	String	A string identifying the view used to define the fragment.

Table 1. The `get-topic` operation

rights, etc. As there are many ways to decide this, and as the decision strictly speaking does not affect interoperability, the mechanisms for determining this are left undefined.

A set of topics is returned containing, for each queried topic map, all topics matching the parameters. The matching topics are all topics which have one of the URIs in `item` as an item identifier, one of the URIs in `subject` as a subject locator, or one of the URIs in `identifier` as a subject identifier.

The set of topics found can have any cardinality. However, all of the topics found are merged into a single topic (in the returned fragment; the state on the server should not change), as the semantics of the operation is to return a single topic. It follows from this that the identifiers passed in the parameters all identify a single subject, and so even if multiple topics may be found (in the same or in different topic maps) they must necessarily all represent the same subject. A fragment representing the merged topic is returned in the syntax specified in the `syntax` parameter. The default will be to use XTM, but other alternatives are possible, as described in 2.4 on page 12.

A challenge in extracting fragments from topic maps is knowing where to stop. Every topic is defined by means of other topics; its associated topics, topic type, association types, occurrence types, name scopes, and so on are all topics. For each of these topics one might conceivably include just the identity, the identity and the names, or a full fragment, and it is not obvious which of these options best serve the needs of users.

Analysis of the use cases suggests that there are situations where each of these possibilities may be what users want, and in some cases even finer-grained control may be needed. The `view` parameter can be used to specify what should be included in the fragment returned. The requested topic will always be included in full; the view applies to the topics referenced by it. The possible values of this parameter are:

- `stub`: only a single identifier will be included, as defined in [Garshol02]. (This is the default view.)
- `names`: the identity and the names will be included.
- `complete`: complete fragments are included, but topics referenced from these fragments will use the `stub` view.

It is also possible to define custom views with TM-Views [Garshol05] and reference these using the `view` parameter. It is assumed that the views will have been registered with the TMRAP server before the request is received. There is no support for using multiple views when querying multiple topic maps since the client in any case receives a single fragment. That is, to the client there is only one view.

The `get-tolog` operation This operation is used to get an XML document representing the results of a tolog query, either as a fragment or as an explicit representation of the result. The parameters to the operation are shown in table 2.

Parameter	Required?	Repeatable?	Type?	Description
tolog	yes	no	String	A tolog query.
topicmap	yes	no	String	A topic map handle.
syntax	no	no	String	A syntax identifier.
view	no	no	String	View identifier.

Table 2. The `get-tolog` operation

This operation has two modes of operation: if the requested `syntax` is tolog there are no restrictions on the query, and an XML structure giving the actual query results is returned. Otherwise, the query must produce only a single column containing only topics, and all topics in the result are output in the same way as for the `get-topic` operation, except that they are not merged (as there is no implication that all topics found by a single tolog query must represent the same subject).

A request against the Opera topic map requesting tolog syntax using the `stub` view and the following query

```
select $COMPOSER, count($OPERA) from
 composed-by($OPERA : opera, $COMPOSER : composer)
order by $OPERA desc limit 2?
```

would produce the following result:

```
<result xmlns:x="http://www.topicmaps.org/xtm/1.0/"
        xmlns:l="http://www.w3.org/1999/xlink">
  <head>
    <column>COMPOSER</column>
    <column>OPERA</column>
  </head>
  <body>
    <row>
```

```
      <value>
        <x:subjectIndicatorRef
           l:href="http://en.wikipedia.org/wiki/Verdi"/>
      </value>
      <value>28</value>
    </row>
    <row>
      <value>
        <x:subjectIndicatorRef
           l:href="http://en.wikipedia.org/wiki/Mascagni"/>
      </value>
      <value>16</value>
    </row>
  </body>
</result>
```

When the syntax is tolog, the fragments within the `value` elements are included using XTM.

The RELAX-NG schema [ISO19757-2] for tolog query results is:

```
start =
  element result {
    element head {
      element column { text }*
    },
    element body {
      element row {
        element value { any }*
      }*
    }
  }

# using wildcard here as there are many alternatives for the
# fragments, and specifying them  all is complex
any = (text | element * { anyatt*, any })*
anyatt = attribute * { text }
```

The `add-fragment` operation This operation is used to import a topic map fragment into the repository. The parameters to the operation are shown in table 3 on the next page.

The fragment is imported into the identified topic map. Usually this will be used to create a single topic, but the operation is deliberately not restricted to only this. Implementations will most likely have limits on the sizes of the fragments they accept.

Parameter	Required?	Repeatable?	Type?	Description
syntax	yes	no	String	A string identifying the syntax used in the fragment.
fragment	yes	no	String	A fragment representing part of a topic map.
topicmap	yes	no	String	A topic map handle.

Table 3. The `add-fragment` operation

Formally, the operation deserializes the received fragment into a TMDM instance [ISO13250-2], then merges that instance into the TMDM instance identified in the `topicmap` parameter, using normal TMDM merging rules.

The `delete-topic` operation This operation is used to delete a topic from a topic map in the repository. The parameters to the operation are shown in table 4.

Parameter	Required?	Repeatable?	Type?	Description
item	no	yes	URI	An item identifier of the sought topic.
subject	no	yes	URI	A subject locator of the sought topic.
identifier	no	yes	URI	A subject identifier of the sought topic.
topicmap	no	yes	String	A topic map handle.

Table 4. The `delete-topic` operation

All topics in the selected topic map(s) which match the parameters are deleted. Deleting a topic means removing all its base names, variants, and occurrences, as well as all associations in which it plays a role. The topic will also be removed wherever it is used as a scope or type, but the scoped and typed topic map constructs are left undeleted.

The rationale for deleting all associations is that after removing one role from the association unary associations are invalid, while binary associations are meaningless. Associations of higher arities might still be meaningful, but in the interest of simplicity they are treated the same way. Several years of experience with this operation (in an API, admittedly) suggests that in practice this works very well.

The `get-topic-page` operation This operation is used to ask the server whether it has any pages for a specific topic. The pages in question might be pages in some Topic Maps application that can display the topic to a user (known as "view pages"), or pages where the user can edit the topic (known as "edit pages"), or other kinds of pages.

The `get-topic-page` operation is really meant to satisfy the portal integration use case mentioned above, and in particular the scenario described in

[Pepper04] as "VISIT", where one portal dynamically links to the topic page for the same topic in another portal.

The parameters to the operation are shown in table 5 on the next page.

Parameter	Required?	Repeatable?	Type?	Description
item	no	yes	URI	An item identifier of the sought topic.
subject	no	yes	URI	A subject locator of the sought topic.
identifier	no	yes	URI	A subject identifier of the sought topic.
topicmap	yes	no	String	A topic map handle.
syntax	yes	no	String	A string identifying the syntax to be used in the response.

Table 5. The `get-topic-page` operation

The response from the request is a topic map describing the structure on the server. The response must contain at least what is described here, but the decision to return a topic map rather than some custom XML format means that the operation is inherently extensible. In the following, the prefix `rap` is to be understood as the subject indentifier namespace `http://psi.ontopia.net/tmrap/`.

The response must include the following:

- A topic of type `rap:server`, representing the server.
- One topic of type `rap:topicmap` for each topic map in which one or more matching topics were found. Each topic map must have exactly one occurrence of type `rap:handle` containing the topic map handle. Each topic map must have a `rap:contained-in` association to the server it's hosted on.
- A single topic containing the results of merging all matching topics. Only identifiers and names need be included.
- For each view and edit page for this topic on the server a topic of type `rap:view-page` or `rap:edit-page`, with the URI of the page as the subject locator of the topic. Each page must also have a `rap:contained-in` association to the topic map it is rendered from.

Note that if no topics are matched the response will contain only the server topic.

An example might help clarify this. The result of asking Ontopia's online demo server (once it's set up) for the topic "Japan" would give the following result (when querying the `i18n.ltm` topic map) in TM/XML.

```
<topic-pages xmlns="http://psi.ontopia.net/tmrap/"
             xmlns:tm="http://psi.ontopia.net/xml/tm-xml/"
             xmlns:iso="http://psi.topicmaps.com/iso13250/"
             xmlns:oasis="http://psi.oasis-open.org/iso/3166/#">

  <server id="online-demo">
    <iso:topic-name>
```

```
      <tm:value>Ontopia Omnigator online demo</tm:value>
    </iso:topic-name>
  </server>

  <topicmap id="i18n.ltm">
    <iso:topic-name>
      <tm:value>Scripts and languages</tm:value>
    </iso:topic-name>
    <handle datatype="http://www.w3.org/2001/XMLSchema#anyURI"
       >i18n.ltm</handle>
    <contained-in role="containee" otherrole="container"
                  topicref="online-demo"/>
  </topicmap>

  <oasis:country>
    <tm:identifier>http://psi.oasis-open.org/iso/3166/#392</tm:identifier>
    <iso:topic-name>
      <tm:value>Japan</tm:value>
    </iso:topic-name>
  </oasis:country>

  <view-page id="p1">
    <tm:locator>http://www.ontopia...?tm=i18n.ltm&id=japan</tm:locator>
    <contained-in role="containee" otherrole="container"
                  topicref="i18n.ltm"/>
  </view-page>
</topic-pages>
```

The `add-type-listener` operation This operation is used to register a client to receive callbacks for all updates to topics of a specific type. The parameters to the operation are shown in table 6.

Parameter	Required?	Repeatable?	Type?	Description
item	no	yes	URI	An item identifier of the sought topic.
subject	no	yes	URI	A subject locator of the sought topic.
identifier	no	yes	URI	A subject identifier of the sought topic.
topicmap	yes	no	String	A topic map handle.
client	yes	no	Handle	The client handle is defined by the binding.
syntax	no	no	String	A string identifying the syntax to be used in notifications.

Table 6. The `add-type-listener` operation

All topics matching the parameters are found in the identified topic map. It is an error if this is not exactly one topic. This means that one will get an

error message if the topic type is not found on the server, or if what the client considered to be one topic is more than one topic to the server.

Every time a topic that is an instance of this type is created, modified, or deleted the corresponding client operation is triggered on all clients. Registrations are persistent until explicitly removed.

The `syntax` parameter is used by the client to indicate what syntax it would like to receive notifications in. All `topic-created` and `topic-updated` notifications must use this syntax. The default is XTM.

The `remove-type-listener` operation This operation is used to unregister a client that has already registered with the `add-type-listener` request so that update callbacks are no longer received. The parameters to the operation are shown in table 7.

Parameter	Required?	Repeatable?	Type?	Description
item	no	yes	URI	An item identifier of the sought topic.
subject	no	yes	URI	A subject locator of the sought topic.
identifier	no	yes	URI	A subject identifier of the sought topic.
topicmap	yes	no	String	A topic map handle.
client	yes	no	Handle	The client handle is defined by the binding.

Table 7. The `remove-type-listener` operation

All topics matching the parameters are found in the identified topic map. It is an error if this is not exactly one topic. This client is then removed as one of the clients registered to receive callbacks for this topic type. It is an error if this client is not registered previously.

2.3 Client operations

The operations in this section are operations on the client invoked by the server in response to the client registering itself using the `add-type-listener` operation.

The `topic-created` operation This operation is invoked by the server every time a topic of a type which the client has registered itself as a listener for is created. The parameters are shown in table 8.

The fragment provided contains the created topic (in the syntax requested by the client). The interface does not require any specific behaviour from the client in response to the request.

The `topic-updated` operation This operation is invoked by the server every time a topic is updated of a type which the client has registered itself as a listener for. The parameters are shown in table 9.

Parameter	Required?	Repeatable?	Type?	Description
server	yes	no	URI	The URI of the server.
topicmap	yes	no	String	A topic map handle.
fragment	yes	no	String	A fragment representing the created topic.

Table 8. The `topic-created` operation

Parameter	Required?	Repeatable?	Type?	Description
server	yes	no	URI	The URI of the server.
topicmap	yes	no	String	A topic map handle.
fragment	yes	no	String	A fragment representing the updated topic.

Table 9. The `topic-updated` operation

The fragment provided contains the updated topic as it was after the update, in the syntax requested by the client. The interface does not require any specific behaviour from the client in response to the request.

Note that the change to the topic may be to the identifiers, in which case the client may not be able to tell which topic has changed. For this reason the server must include any identifiers removed or added in the update in the notification, but the removed identifiers must be omitted in following update notifications.

The `topic-deleted` operation This operation is invoked by the server every time a topic is deleted of a type which the client has registered itself as a listener for. The parameters are shown in table 10.

Parameter	Required?	Repeatable?	Type?	Description
server	yes	no	URI	The URI of the server.
topicmap	yes	no	String	A topic map handle.
item	no	yes	URI	An item identifier of the deleted topic.
subject	no	yes	URI	A subject locator of the deleted topic.
identifier	no	yes	URI	A subject identifier of the deleted topic.

Table 10. The `topic-deleted` operation

The identifiers given identify the deleted topic to the client. No specific behaviour is required from the client.

2.4 Syntax identifiers

The syntaxes are identified by their MIME types [RFC2045]. The valid alternatives are shown in table 11. If no syntax is specified, the default is to produce an XTM fragment as defined in [Garshol02].

Syntax	MIME type
XTM	application/x-xtm
LTM	text/x-ltm
AsTMa=	text/x-astma
TM/XML	text/x-tmxml
tolog	text/x-tolog

Table 11. Topic map syntax MIME types

The TM/XML syntax is described in [Garshol05].

3 The HTTP binding

An HTTP binding of the TMRAP abstract interface could take several approaches. It could use SOAP [SOAP]; it could take a RESTful approach [Fielding00]; or it could aim for a more straightforward, traditional HTTP approach. As mentioned in the abstract, this paper opts for the last of these, but there are plans to add SOAP support in the future.

3.1 To REST or not to REST

REST is best thought of as a style guide for creating web services, famously defined by [Fielding00]. The argument for not using it in TMRAP is that it recommends using the HTTP methods (GET, PUT, etc) to operate directly on resources exposed on the web. It could be described as object-oriented instead of the traditional procedural approach, where URIs represent resources (or objects) instead of procedures.

It should be quite clear from the operations provided that TMRAP is very much in the traditional camp, and not at all REST-like. The rationale is that, perhaps somewhat perversely, Topic Maps provide no easily addressable isolated resources to expose and operate on. Further, REST makes heavy demands on the underlying HTTP infrastructure, which may not always support everything that is needed very well (URL mapping of complex URLs, obscure HTTP operations, etc etc).

In short, the argument generally put forward for REST is elegance [Barta05], whereas the argument against it in this paper is lack of elegance for this particular purpose, as well as a desire to avoid infrastructure problems.

3.2 The binding itself

The general approach taken by the binding is simple: server and client endpoints are defined using HTTP URIs. Each operation has a separate URI obtained by concatenating the endpoint URI with the operation name. Each parameter becomes a URI query parameter in the traditional `?foo=1&bar=2&baz=3` syntax.

Authentication and access control are not considered part of the HTTP binding, but are provided by the application server itself, using the normal HTTP mechanisms.

It's tempting to map the `syntax` parameter to the `Accept` header in HTTP, but for this to provide any benefit it requires the user to learn the syntax for specifying alternatives, and it requires implementations to do the same. Few HTTP client libraries provide any support for this, and so it seems better to map `syntax` in the same way as the other parameters.

All operations which make modifications (this includes the client operations) must be accessed using the POST method, while all operations which only retrieve information must use GET. (This means that the `fragment` parameter will automatically travel in the request body, as with POST all parameters are transmitted in the body.)

If errors occur, as defined above, the server must return an HTTP response with response code 400 ("Bad Request"). Including an informative error message in the response is encouraged.

4 Related work

Substantial work has already been done in this area, and so the charge might be made against TMRAP that it needlessly proliferates the number of alternative interfaces. To answer this charge we review related work.

The most complete and well-documented alternative Topic Maps interface proposal to date is clearly TMIP[Barta05]. This protocol takes a REST-based approach, and is entirely dependent on TMQL (which is not yet stable). It supports retrieval of fragments in various syntaxes, and updates to selected topics. There is no explicit support for deletion and creation, however, although there are hints as to how these might be achieved. There is no support for events.

An alternative is the Topic Maps Service[NetworkedPlanet05] web service based on SOAP and WSDL. This service provides predefined methods for returning topic fragments by certain criteria (topics by type, topic by id, topic by subject identifier, hierarhices, etc), and also for updates and deletes. There is also support for retrieving fragments by means of TMRQL queries. There is no support for events, and TMRQL is unsuitable for our purposes as it requires the topic maps to be stored in specific SQL databases [Barta05b].

A third alternative is the SPARQL Protocol for RDF[SPARQL], which is based on RDF and the RDF query language SPARQL. The protocol is abstract, and has a standard WSDL binding. The present version only provides support for running SPARQL queries and returning the results.

Other related work is reviewed by [Barta05], of which the most relevant are Shark[Schwotzer04] and [Thompson04]. Shark is designed for mobile handheld units, and so has rather different design considerations. [Thompson04] is interesting, but based on the as-yet unstable TMQL, REST-based, and not described in any detail.

As should be evident, no interface currently exists that, in our opinion, meets all our use cases. In particular, no other interface provides event callbacks. Of the Topic Maps-based proposals, two use TMQL (which is unstable), the third uses TMRQL (which is unsuitable), and the fourth (Shark) is intended for a different environment.

Some protocols [Barta05] provide operations not found in TMRAP that allow clients to get information about the server. Such information might include which topic maps are available, which formats are supported by the server, etc. Such operations have been left out of TMRAP as there is nothing in the use cases to suggest that it would be useful. Introspection operations only seem useful in cases where clients are looking servers up in some form of registry and connecting to them dynamically. However, the use cases all involve interaction between a client and a server already known to the person (or tool) configuring the client, and so support for this does not seem necessary.

5 Conclusion

In this article is presented the design of a Topic Maps web service interface that is based on stable and documented technologies, and which, we believe, satisfies a number of important use cases, and thus opens the possibility for Topic Maps applications that are more open and accessible than what has been seen thus far.

5.1 Further work

The web service interface will be implemented in the commercial Ontopia Knowledge Suite (OKS) over the coming months, and used in a number of different projects. Further revisions will be made if experience with usage in these projects indicate that revisions are needed.

In addition, it is thought that special requests that allow legacy data (XML that is not a Topic Maps syntax, CSV files, etc) to be imported into a topic map may well be needed. That is, clients may wish to add fragments of data to a topic map that is not in any Topic Maps syntax. In these cases, it may be easiest for the server to handle the conversion into Topic Maps, and so special requests may be added that allow legacy data to be imported directly into the server. (One assumes that some form of conversion tool or configuration will already have been installed on the server.)

Finally, it is possible that support will be added for subscription to RSS channels containing topic map update information. Enabling the listener mechanism to support more fine-grained subscription, possibly via tolog queries, is also being considered.

References

[Barta05] *TMIP, A RESTful Topic Maps Interaction Protocol*; Barta, R.; Extreme Markup 2005, Montral, Canada.

http://www.mulberrytech.com/Extreme/Proceedings/html/2005/Barta01/EML2005Barta01.html

[Barta05b] *SQL as TM Query Language? No, thanks!*; Barta, R.; private blog entry, undated. http://topicmaps.it.bond.edu.au/docs/38?style=printable

[Fielding00] *Architectural Styles and the Design of Network-based Software Architectures*; Fielding, R. T.; Doctoral dissertation, University of California, Irvine, 2000. http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm

[Halevy05] *Enterprise Information Integration: Successes, Challenges and Controversies*; Halevy, A., Ashish, N., Bitton, D., Carey, M., Draper, D., Pollock, J., Rosenthal, A., Sikka, V.; Proceedings of SIGACM-SIGMOD '05 Baltimore, USA; 778-787. http://www.cs.washington.edu/homes/alon/files/eiisigmod05.pdf

[Garshol02] *XTM Fragment Interchange 0.1*; Garshol, L. M.; Ontopia Technical Report 2002-09-23. http://www.ontopia.net/topicmaps/materials/xtm-fragments.html

[Garshol02b] *Topic maps in content management – The rise of the ITMS*; Garshol, L. M.; Proceedings of XML 2002, IDEAlliance, Baltimore, USA; 2002-12-08. http://www.ontopia.net/topicmaps/materials/itms.html

[Garshol05] *TM/XML – Representing Topic Maps in XML*; Garshol, L. M., Bogachev, D.; forthcoming, to be published in proceedings of TMRA'05.

[Garshol05b] *tolog – a topic maps query language*; Garshol, L. M., forthcoming, to be published in proceedings of TMRA'05.

[ISO13250-2] ISO 13250-3: Topic Maps – Data Model; International Organization for Standardization; Geneva. http://www.isotopicmaps.org/sam/sam-model/

[ISO19757-2] ISO 19757-2: Document Schema Definition Languages (DSDL) – Part 2: Regular-grammar- based validation – RELAX NG; International Organization for Standardization; Geneva. http://www.y12.doe.gov/sgml/sc34/document/0362_files/relaxng-is.pdf

[Moore03] *Semantic Web Servers*; Moore, G.; Extreme Markup 2003, Montral, Canada. http://www.ontopia.net/topicmaps/materials/semantic-web-servers.ppt

[Moore04] *Topic Maps Remote Access Protocol*; Moore, G.; 2004-04-06, Ontopia. http://www.jtc1sc34.org/repository/0507.htm

[NetworkedPlanet05] *Topic Map Web Services*; Moore, G., Ahmed, Kal; NetworkedPlanet. Available on 2005-08-12 from http://www.networkedplanet.com/technology/webservices/intro.html

[Pepper04] *Seamless Knowledge–Spontaneous Knowledge Federation using TMRAP*; Pepper, S., Garshol, L. M.; Extreme Markup 2004, Montral, Canada. http://www.ontopia.net/topicmaps/materials/ Seamless+Knowledge+with+TMRAP.ppt

[RFC2045] *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*; Freed, N., Borenstein, N.; IETF RFC 2045; November 1996; http://www.isi.edu/in-notes/rfc2045.txt

[SOAP] *SOAP Version 1.2 Part 1: Messaging Framework*; Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., Frystyk Nielsen, H.; W3C Recommendation; 24 June 2003. http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[Schwotzer04] *Shark - a System for Management, Synchronization and Exchange of Knowledge in Mobile User Groups*; Schwotzer, T., Geihs, K.; Technical Report, Intelligent Networks and Management of Distributed Systems, TU Berlin; http://ivs.tu-berlin.de/ thsc/Shark_IKnow.pdf

[SPARQL] *SPARQL Protocol for RDF*; Grant Clark, K.; W3C Working Draft 27 May 2005; http://www.w3.org/TR/rdf-sparql-protocol/

[Thompson04] *Scalable, document-centric addressing of semantic stores using the XPointer Framework and the REST architectural style*; Thompson, B., Moore, G., Parsia, B., Bebee, B. R.; Extreme Markup 2004, Montral, Canada. http://www.mulberrytech.com/Extreme/Proceedings/html/2004/Thompson01/EML2004Thompson01.html