TMAPI-implementation for shared accessed topicmaps

UNIVERSITÄT LEIPZIG

Martin Krüger, Jan Hellich

Fakultät für Mathematik und Informatik,
Universität Leipzig

Abstract

Since topicmaps are an efficient way to organise information it is important to enable concurrent and shared access to this information for optimal use.
This project is a proof-of-concept. Therefor we chose to use unusual ways to reach this goal.
We use a 3-tier approach to divide the tasks of interfacing the user application, processing the topicmaps (inserting, finding and deleting information) and persistent storage of the information.
The pluggable part for applications implements the TMAPI and is slim in size and function. It communicates with an application server (Glassfish) which provides Webservices (TM-WS). These Webservices do all operations on a topicmap which itself is located on a SQL-server (MySQL).

Considerations

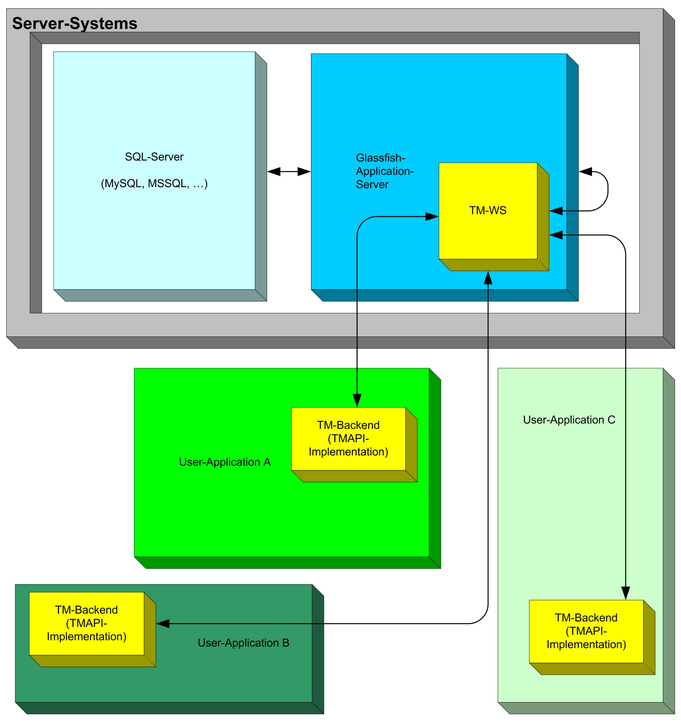
In order to enable multiple users access to the topicmap we found the following items important to be considered.:

- centralise the topic map storage
- connect the client to the topic maps over a network
- minimise hardware requirements for clients as much as possible
- base the project on freely available technologies

Since this is only proof of concept we only implemented XTM 1.0 features. We are aware that this is a limitation. In case of further interest in this matter it should be easily possible to implement missing features in future versions.

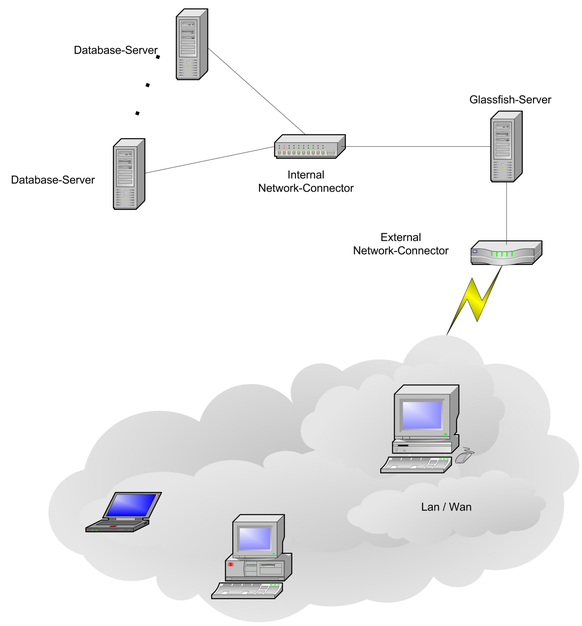
Technical environment

To meet the criteria to minimise client hardware requirements as much as possible we chose a 3-tier approach. The program which uses the TMAPI-interface (implemented in TM-Backend) accesses an application server which runs webservices. These webservices contain the logic necessary to insert, find and delete information in a topicmap. The whole topicmap is stored in a SQL-server. This should provide enough processing speed and data security.



(a) Schema of program design

The programming language of choice is Java for its wide range of supported platforms. The application server used is the Glassfish application server, a reference implementation of SUN. As database we chose MySQL which is also freely available.



(a) Prossible network structure

Because of the architecture of the application server all components a exchangeable. The design provides system administrators with a wide range of choices for database servers, application servers and even operating systems. Without recompile and a minimal amount of reconfiguration of the application server the administrator could change from a MySQL-server to an Oracle based system for example. In short.: If you prefer another database you can use it. If you want to replace the client component (TM-Backend) but want to use the webservices you can do so. (In case you have other client programs accessing the webservices.) In theory it is even possible to loadbalance the webservices in a cluster of application servers.

Code Example

This code snippet shows the TM-Backend side (the TMAPI implemenation). It is the function to retreave all types of a topic in the class “Topic”.

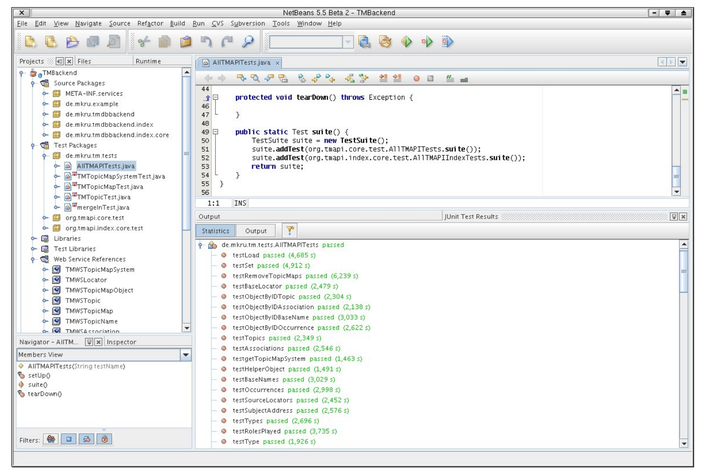
```
public Set getTypes() {
    Set s = new HashSet();
    List l = null;

    try {
        l = WSCachingServiceLocator.getInstance().getTopic()
                    .getTypes(this.getIntObjectID());
    } catch (Exception ex) {
        ex.printStackTrace();
        throw new TMAPIRuntimeException("A connection " +
                "error has occured.");
    }
    int i;
    for (Iterator it = l.iterator(); it.hasNext();) {
        i = Integer.parseInt(it.next().toString());
        s.add(new TopicImpl(i, this.getBelongingTopicMapSystem()));
    }
    l = null;
    return(s);
}
```

This code snippet is the equivalent to thecode presented above. Its functionality performs database operations and returns all types of the specified topic.

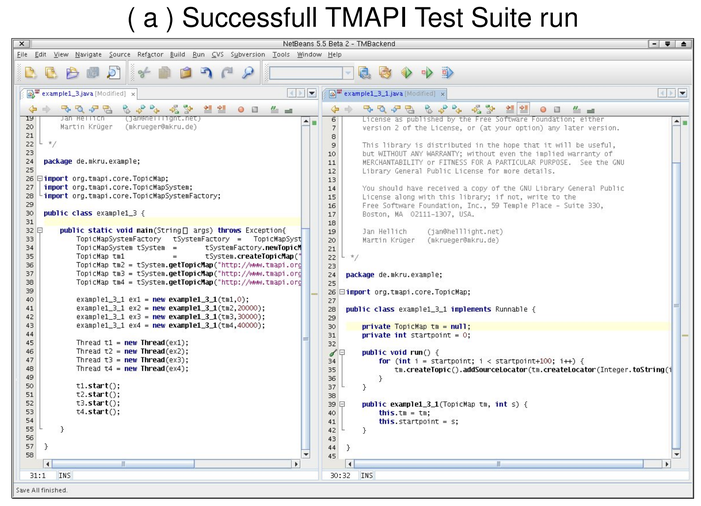
```
/**
 * Web service operation
 */
@WebMethod
public List getTypes(@WebParam(name = "intTopicID") int intTopicID) {
    Connection conn = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;

    List value = new ArrayList();

    try {
        conn = tmws.getConnection();

        pstmt = conn.prepareStatement("SELECT topicTypeID "+
                    "FROM topicTypes "+
                    "WHERE topicID=? ORDER BY topicTypeID ASC;");
        pstmt.setInt(1, intTopicID);

        rs = pstmt.executeQuery();

        while (rs.next()) {
            value.add(rs.getInt("topicTypeID"));
        }

        [...]

    return value;
}
```

Perfomance

Since this project had limited resources it was not possible to run a wide variety of benchmarks.
We wrote a few testcases we belief to be common in using topicmaps. (The source code is available in the program package.)
First we checked how fast certain database intensive operations are. Then we stressed the system with many concurrent connections and messured what the limit of our development machine is.
But do to the complex nature of this architecture we were unable to produce reliable numbers. It seems to us that the nature of the network link has a huge impact to the overall performance of the application. Not the link speed is the driving factor but rather the link latency.
We chose not to present numbers on this poster because there are aspects of influences in these test setting which we not fully understand. We guess that this is partually the experimental nature of the Glassfish application server.



(a) Successfull TMAPI Test Suite run



(b) Example test program

Grafic a) shows the successfull run of the TMAPI Test Suite. In grafic b) a simple example is shown which simulates 4 concurrent connections.

Outlook

Right now the code is proof-of-concept and should in no circumstances be used in any production process. We would not recommend otherwise until further stability and performace tests are made. The most interesting question is how well different databases and application server play together. This leads to the question of stability, hardware requirements and procurement costs.
In case this concept proofs that its worth further development additional features like XTM 1.1 could be implemented.

References

- http://www.tmapi.org/ - TMAPI (implemented interface)
- http://tinytim.sf.net/ - TinyTIM (Memory based TMAPI implementation)
- http://www.mysql.com/ - MySQL server
- http://glassfish.dev.java.net/ - Glassfish application server
- http://www.http://tm4j.org/ - another topicmap engine
- http://www.topicmap.com/
- http://www.topicmaps.org/
- http://www.ontopia.net/