Binding the Simple Query Interface

Nhu Van Nguyen and David Massart

European Schoolnet, rue de Trèves 61, B-1040 Brussels, Belgium.
{nhuvan.nguyen,david.massart}@eun.org

Abstract. The Simple Query Interface (SQI) is a query transport standard that is becoming widely used within the e-learning community. Thanks to a common WSDL binding, SQI-compliant repositories of learning resources can query each other. This common binding directly maps the SQI specification. As a consequence, many remote calls (known to present an important latency) are necessary to query a repository. This paper proposes to reduce the number of remote calls necessary to query an SQI repository by applying the remote façade pattern to SQI bindings. A WSDL, an HTTP, and a JMS binding are proposed to illustrate this approach.

Keywords: SQI, WSDL binding, learning resource repository, interoperability, performance, remote façade.

1. Introduction

The Simple Query Interface (SQI) is a standard Application Program Interface (API) used to query repositories of learning resources. SQI serves as a universal interoperability layer for educational networks. In order to be truly interoperable, repositories that use SQI need also to share a common binding that ensures compatible implementations of the standard. A "Common SQI WSDL" was thus proposed [4]. This common binding is now in use in federations of learning resource repositories such as GLOBE [3].

The current version of the common SQI WSDL binding is a direct mapping of the SQI specifications. Each SQI method corresponds to a web service method. As a consequence, multiple remote method invocations are required when a system queries a remote repository. These remote procedure calls are slow and result in degraded performance [2].

Currently, SQI implementations address this performance issue by defining default values. The use of default query parameters such as query language, results format, maximum duration, or maximum query results permits to limit the number of remote calls. However, the use of default parameters implies to renounce, at least partly, to the flexibility offered by the SQI specification. As a consequence, this binding force developers to trade off flexibility for performance.

In this paper we propose to apply the remote façade pattern to the design of SQI bindings, which permits to reduce the number of remote calls by passing more data with each call. This technique was used to bind SQI to WSDL, HTTP, and JMS.

This paper is organized as follows: Section 2 provides an overview of the Simple Query Interface (SQI). Section 3 briefly describes the remote façade pattern and applies it to SQI. Section 4 proposes an SQI WSDL binding based on the remote façade pattern and compares it to the Common SQI binding. Finally, an HTTP and a JMS bindings are proposed in Section 5.

2. Overview of the Simple Query Interface (SQI)

The Simple Query Interface is an Application Program Interface (API) for querying repositories of learning resources. It is characterized by its relative simplicity and its flexibility that allow it to be easily deployed in a large variety of settings. SQI is based on a session management concept that permits to separate authentication issues from query management[1]. It is neutral in terms of query languages and result formats, supports both, a synchronous and an asynchronous query mode and can be implemented as a stateless or stateful service.

Considering two repositories sharing at least a common query language and a common metadata format, the following steps are necessary to enable one repository (referred as the source of the query) to query the other (referred as the target of the query) using SQI [5]:

- The source selects one of the query languages available at the target (e.g., XQUERY – It is possible to skip this step when a default query language is proposed by the target),
- The source selects one of the result formats available at the target (e.g., the IEEE Learning Object Metadata binding – Here also it is possible to skip this step when a default result format is proposed by the target),
- The source sends a query in the selected query language,
- Depending on the query mode selected, the target provides the result of the query in the selected format either as the return value of the call used to send the query (synchronous mode) or by calling one or more times a query result listener implemented by the source (asynchronous mode). The latter mode is much more robust and enables SQI to be used as the front-end interface of a federated search since it is not necessary to wait for the end of the initial query before returning the first results.

The API itself is depicted on the class diagram of Fig. 1. It consists of ten methods (1 for the source and 9 for the target) that can be grouped into three categories: query management, synchronous query management, and asynchronous query management.

[1] Session management is not part of the SQI specification itself. This allows SQI to be combined with the session management mechanism that is considered as the most appropriate for each context of utilization.

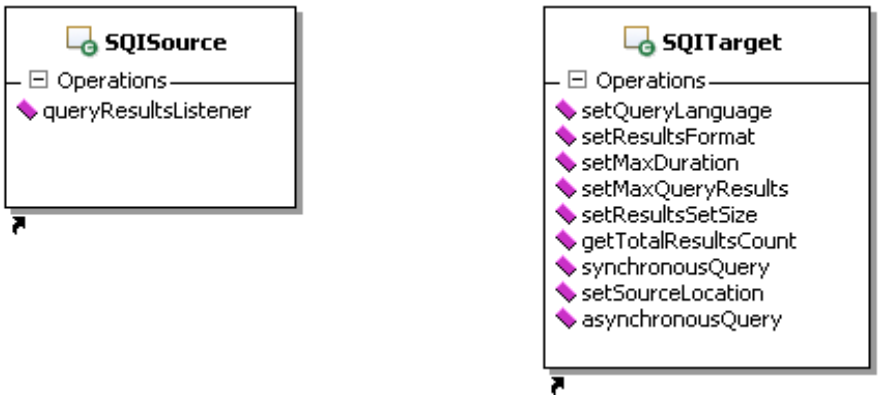

Figure 1. Class diagram of the Simple Query Interface.

The query management methods permit the configuration of query parameters such as the query language (*setQueryLanguage*), the format of the results (*setResultsFormat*), the maximum number of results returned (*setMaxQueryResults*), and the duration of a query (*setMaxDuration*).

In a synchronous query, query results are returned as the result of a query call (*synchronousQuery*). Additional methods permit the choice of the number of results returned by a call (*setResultsSetSize*) and to know the total number of results of a query (*getTotalResultsCount*).

In an asynchronous query, query results are sent by the target to the source of the query by calling a listener implemented by the source (*queryResultsListener*). This implies that the source has to indicate the location of the listener to the target (*setSourceLocation*) before sending an asynchronous query (*asynchronousQuery*).

As it can be seen on Figure 1, SQI design is based on the "Command-Query Separation Principle". This principle states that every method should either be a command that performs an action, or a query that returns data to the caller, but not both. In addition, in order to make the interface easily extensible an approach, minimizing the number of parameters of the various methods rather than the number of methods was adopted. Variations of the interface (e.g., a separation between common query schema and common results format), can easily be introduced by adding a new function (e.g., setSupportedQuerySchema) while no change in the already implemented methods is needed. Hereby, backwards compatibility can be more easily maintained [6].

As a result, SQI provides a set of fine-grained methods for managing queries. Each method is responsible for a single, fairly small, and atomic piece of functionality, which simplifies programming and provides better abstraction from the object internals and thereby increases potential for reuse.

This being said, the use of fine-grained method also implies invoking more methods to perform a high-level task. In the case of SQI, the cost of these extra function calls can be severe because these methods are usually invoked across process and network boundaries.

3. An SQI Remote Façade

The "remote façade" pattern [2] provides a coarse-grained layer in front of fine-grained objects to improve efficiency over a network. The role of the remote façade consists of translating coarse-grained methods onto the internal fine-grained objects. This permits to reduce network traffic overhead by making as few remote method invocations as possible.

We propose to apply the remote façade pattern to SQI by introducing an SQI Target remote façade as described on the class diagram of Figure 2 (we keep the other elements of SQI unchanged). This SQI Target Remote façade consists of 4 methods: setQueryConfiguration, synchronousQuery, asynchronousQuery, and getTotalResultsCount.

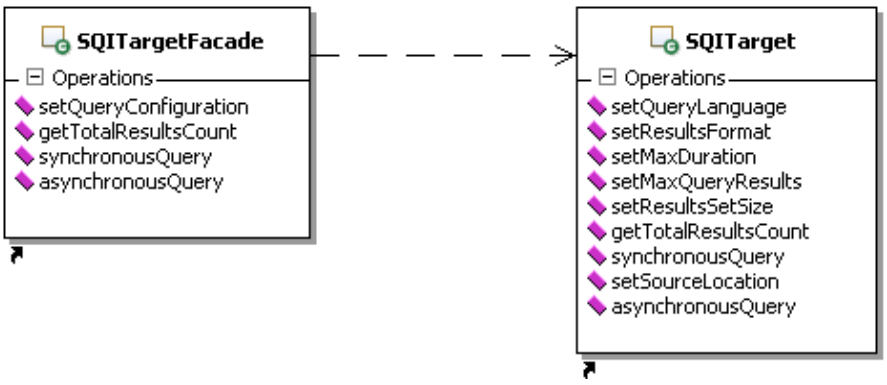

Figure 2. SQITargetFaçade to SQITarget

Each SQI Target remote façade method corresponds to a category of SQI Target methods:

- setQueryConfiguration corresponds to the four query management methods: setMaxQueryResults, setMaxDuration, setQueryLanguage, and setResultsFormat;
- synchronousQuery corresponds to two synchronous query methods: setResultsSetSize and synchronousQuery;
- getTotalResultsCount, the third synchronous method remains unchanged since this method is a query that is used as an alternative to synchronous query.
- asynchronousQuery corresponds to the four query management methods: setMaxQueryResults, setMaxDuration, setQueryLanguage, setResultsFormat and the two asynchronous query methods: setSourceLocation and asynchronousQuery.

We hesitated to merge remote façade methods setQueryConfiguration and synchronousQuery (as we did with remote façade method asynchronousQuery). Finally, we renounced to do it to avoid having to repeat query parameters when method synchronousQuery is called several time in order to get the different chunks of results for a given query.

4. SQI Remote Façade WSDL

The common SQI WSDL binding consists of four WSDL files:

- sqiFault.wsdl describes the SQI Fault (i.e., the fault mechanism associated with SQI);
- sqiSessionManagement.wsdl is provided as an example of possible target session service;
- sqiTarget.wsdl describes the Target SQI interface; and
- sqiSource.wsdl describes the Source SQI interface.

As a direct mapping of the 9 SQI Target methods, the current sqiTarget.wsdl does not address performance issues. Each of the 9 methods corresponds to a web service method. Depending on the foreseen scenario this can lead to intensive interactions between target repositories and the remote systems that query them (many remote method invocations could be required). Such inter-process communications are expensive and the use of multiple remote calls usually results in degraded performance: data have to be marshaled, security may need to be checked, and packets need to be routed through switches [2].

SQI repositories participating in federations (like GLOBE) usually agree on using default values for query parameters such as query language, results format, maximum duration, or maximum query results to limit the number of remote calls necessary during a query. However, this solution does not work for repositories that support multiple query modes, query languages and query results and take advantage of SQI be able to participate in different federations based on different agreements.

To solve the performance issue without renouncing to the flexibility offered by the SQI specification, we propose to replace the current sqiTarget.wsdl by a WSDL binding of the SQI target remote façade presented in Section 3[2]. Files sqiFault.wsdl, sqiSession.wsdl, sqiSource.wsdl remains unchanged.

The sequence diagrams of Figure 3 present typical usages of the proposed SQI Target façade in synchronous and asynchronous modes when default values cannot be used. Both diagrams assume that a session is already established between the source and its targets.

In synchronous mode (left side of Figure 3), the setQueryConfiguration method is used to configure the parameters of the query before submitting it to the target. Then, The method getTotalResultsCount is called to get the total number of results matching the query. Finally, the method synchronousQuery is called, returning a set of metadata records matching the query. Using the proposed binding, this scenario is achieved with 4 remote calls (to the SQI remote façade) instead of the 8 required by the common SQI WSDL binding.

In asynchronous mode (right side of Figure 3), once a session is established, a unique remote call of method asynchronousQuery is sufficient instead of the 6 that were necessary when using common SQI WSDL binding.

[2] We have chosen to use a long list of parameters with sample types instead of using Data Transfer Objects (DTO) to improve the interoperability between systems based on different technologies (e.g., .NET, Java, PHP).

In both query modes, the proposed WSDL binding dramatically reduces the number of remote calls necessary to query an SQI repository while keeping SQI richness and flexibility.

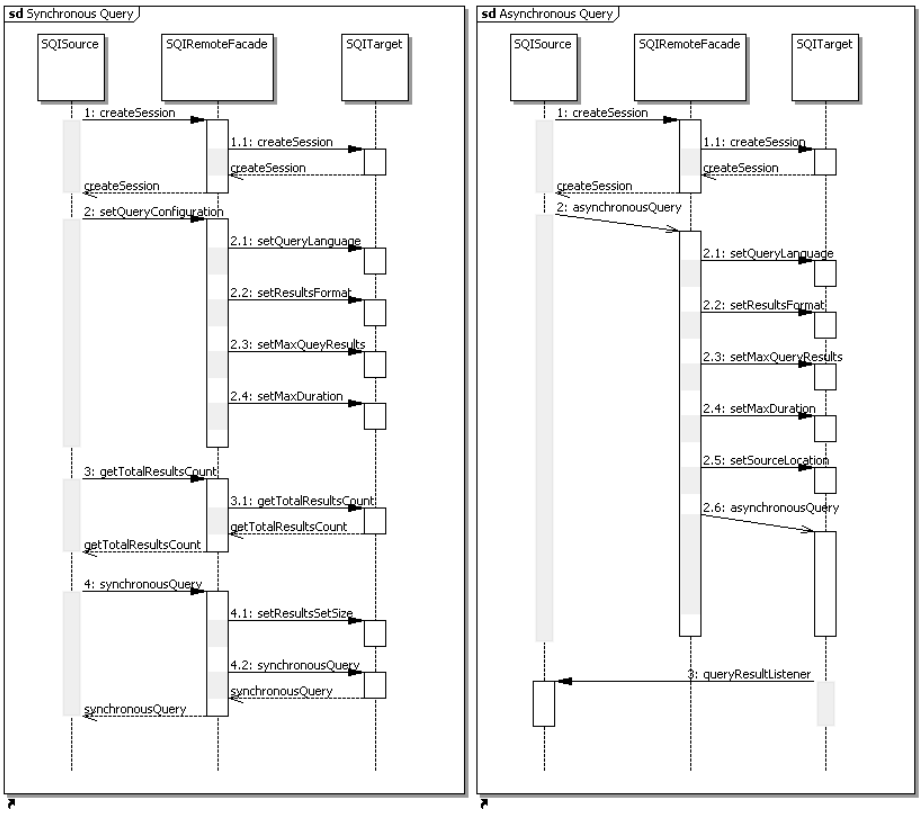

Figure 3. Sequence diagrams of typical usages the SQI Target remote façade in synchronous (left diagram) and asynchronous (right diagram) modes.

Implementing support for the proposed SQI Target remote façade WSDL on MINOR [8] (an open source repository developed by the eMapps.com project [9] that already supports the old binding) proved to be trivial[3]. On this repository, the new binding permits to reduce query time by an average (each scenario was tested one hundred times) of:

- 545.50 milliseconds in the synchronous query mode scenario of Figure 3 (reducing the number of remote calls from 7 to 3 and the overall query time of 56.82 %) and
- 668.73 milliseconds in the asynchronous query mode scenario (reducing the number of remote calls from 6 to 1 and the overall query time of

[3] Note that supporting the new SQI Target remote façade WSDL in addition (instead of in replacement) of the common SQI WSDL binding allows us to offer an improved service while maintaining the compatibility with SQI sources that do not support the new binding yet.

70.29 % - The time necessary to actually process the query explains that the query time diminution is not proportional to the diminution of the number of remote calls).

The proposed SQI Target remote façade WSDL can be found on sourceforge.net at http://sqi-wsdl.cvs.sourceforge.net/sqi-wsdl/wsdl/sqi-1.0%2Bremote-facade/.

5. Other Bindings

In this section, we propose to use the remote façade defined in Section 3 to bind:

- The synchronous part of the SQI Target to an HTTP REST service and
- Its asynchronous part to a Java Message Service (JMS).

5.1 SQI HTTP REST-Style Service

The principle is simple and its implementation is straightforward. All synchronous methods of the SQI façade (i.e., createSession, setQueryConfiguration, getTotalResultsCount, and synchronousQuery) are mapped to an arbitrary URI and its parameters. The first parameter "verb" corresponds to the SQI façade synchronous method to be called. The other parameters correspond to the method parameters. For example, the following URL can be called to send a synchronousQuery:

http://sqiRepository.org/sqi?verb=synchronousQuery&resultSetSize=100&sessionId=session1&startResultIndex=1&queryStatement=water

Similar URLs are used for calling the three other methods.

Note that by using default values, it is possible to get query results in one call which avoids the need of a session Id.

Clients can choose to call such a service by using POST and GET method. Its main advantages are its light-weight (it does not require a lot of extra xml markup) and its ease to build (no toolkit is required).

5.2 SQI JMS Binding

Here also, the binding is relatively straightforward. The session mechanism associated to SQI is used to establish a connection with the relevant JMS destinations (e.g., a JMS topic used to broadcast a federated search). Then, a JMS client exposing the SQI Target interface can be used to send queries as described on Figure 4. All query parameters including the query itself are stored in a Data Transfer Object (DTO) that is loaded into a JMS message. Then, the message is published on a JMS topic where it can be received by all the repositories that have subscribed to this topic. Once a query message is received by a repository JMS client, information is extracted from its DTO to call all the individual SQI methods of the repository.

With this binding, a unique remote call is sufficient to query multiple repositories. Similarly, query results are returned by loading them in a JMS message that is re-

turned to a JMS. This technique is currently used by the LIMBS brokerage system [1].

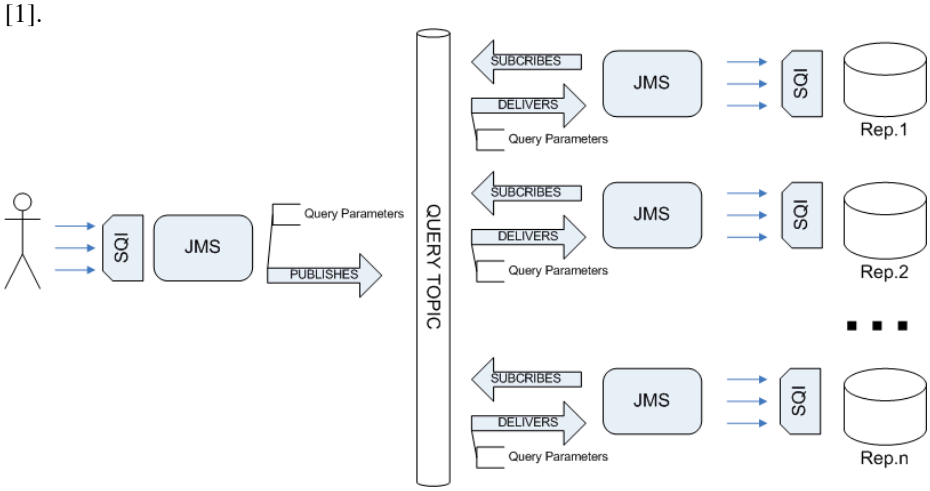

Figure 4. JMS binding for SQI

6. Conclusion

In this paper, we propose to enhance SQI performance by applying the remote façade pattern to its bindings. We present a remote façade for the SQI target interface that we use to design a WSDL, an HTTP, and a JMS SQI binding. These bindings permit to minimize the number of remote calls necessary to query an SQI repository, which dramatically reduces the time necessary to query this repository while preserving SQI flexibility.

The SQI remote façade presented here is an attempt to provide a more efficient binding for SQI. It can certainly be improved. We propose it as a starting point for a discussion with all interested parties that should ideally lead to a replacement for the current common SQI WSDL binding.

Acknowledgment

The work presented in this paper is partially supported by the European Commission under the Information Society Technologies (IST) program of the 6th FP for RTD – as part of the EMAPPS.COM project, contract IST-28051. The authors are solely responsible for the content of this paper. It does not represent the opinion of the European Commission, and the European Commission is not responsible for any use that might be made of data appearing therein.

References:

[1] Colin, J.-N. and Massart, D. (2006) LIMBS: Open source, open standards, and open content to foster learning resource exchanges. In Kinshuk; Koper, R.; Kommers, P.; Kirschner, P.;

Sampson, D. and Didderen, W. (Eds.), Proc. of *The Sixth IEEE International Conference on Advanced Learning Technologies*, ICALT'06, pages 682-686. IEEE Computer Society, Los Alamitos, California.
[2] Fowler M. (2003) "Patterns of Enterprise Application Architecture". Addison Wesley.
[3] The Global Learning Objects Brokered Exchange (GLOBE) website. Last accessed 15 August 2007 at http://globe.edna.edu.au/globe/go.
[4] The Common SQI WSDL Binding project web site. Last accessed 15 August 2007 at http://sqi-wsdl.sourceforge.net/.
[5] D. Massart (2006) A "simple query interface" adapter for the discovery and exchange of learning resources. International Journal on E-Learning (IJEL), 5(1):151-159. Special Issue: Learning Objects in Context.
[6] Simon, B.; Massart, D.; Van Assche, F.; Ternier, S. and Duval, E. (Eds.) (2005) A simple query interface specification for learning repositories. CEN Workshop Agreement (CWA 15454).
[7] European Schoolnet (2007) Learning Resource Exchange (LRE). Last accessed 15 August 2007 at http://lre.eun.org.
[8] The MINOR repository website. Last accessed 15 August 2007 at http://minor.sourceforge.net.
[9] The eMapps.com project website. Last accessed 15 August 2007 at http://emapps.com/.