An Open Distributed Middleware for the Semantic Web

Reto Krummenacher, Daniel Blunder, Elena Simperl, Michael Fried
(Semantic Technology Institute, University of Innsbruck, Austria
firstname.lastname@sti2.at)

Abstract: The emergence of the Web2.0, Web services and the Semantic Web, have changed the WWW to an open interactive market. Heterogeneous and distributed information prosumers have to integrate data and to coordinate activities. This calls for novel middleware solutions that are compliant with the fundamental principles of the Web: persistent publication, scalability and asynchrony of communication. In earlier work we presented a semantic space-based approach, as answer to this challenge. Data is no longer queried directly via endpoint locators, but exposed via a space that offers a unified and virtual view of diverse databases. The space is accessed by use of a simple set of operations that have SPARQL-like constructs at the basis for semantic matching. In this paper we return to the proposed semantic spaces and focus on architectural issues, and describe an implementation with evaluation.

Key Words: Semantic Space, Knowledge Coordination, Middleware

Category: C.2.4, H.3.5, M.6

1 Introduction

Knowledge in the form of semantic data is becoming ubiquitous on the Internet. To access and use this knowledge, exposed in various ways in a multitude of platforms, end-user applications need an integration middleware that provides a loosely-coupled virtualization of the underlying technical complexity and the distributedness of the knowledge sources, e.g. repositories or Web services.

In earlier work we presented for a middleware approach at the intersection of tuplespace [Gel85], or blackboard-style computing [Eng88] and the Semantic Web, as answer to the challenges of large-scale data management and knowledge-intensive service computing on the (Semantic) Web [FKS+07, SKN07]. Our middleware proposal, a semantic spaces platform, is called “Triple Space” and is tailored to the management of information that is formalized using Semantic Web representation languages, to reason about the information, and to coordinate its exchange among distributed entities and data stores that process and maintain the information. Triple spaces have application to the Semantic Web in that they realize open knowledge spaces where (machine-interpretable) data can be persistently published, shared, and coordinated. The core principles behind this approach have been successfully demonstrated in a wide range of space-based systems applied to solve communication and coordination issues in areas which are highly relevant to (Semantic) Web-based systems: open distributed systems, workflow execution, XML middleware or self-organization.
In this paper we return to the proposed Triple Space middleware with a focus on architecture and implementation in large scale, open and distributed Web environments. We introduce an implementation called tsc++ that delivers the core functionality to manage distributed spaces. We evaluate the work and show that already our simplistic realization provides a scalable approach to semantic middleware. The core notions of semantic spaces are shortly reconsidered in Section 2. Section 3 describes the architecture, whilst its technical realization and evaluation are presented in Section 4. This paper does not present P2P research, but we only apply an existing P2P framework to showcase our approach. Related work is in Section 5 and we conclude with Section 6.

2 Conceptual Model

A space can conceptually be seen as a database that delivers communication and coordination services. Tuples refer to records in a non-normalized database table without primary keys, the publish operation relates to an insert, the retrieval operation to a select statement. Indeed, many tuplespace systems use a database to ensure durability, and distributed systems are built by allowing an application to access several spaces (databases). A space acts as an abstraction layer that allows to coordinate distributed data sources and services invisible for the application developers. Applications must only take one virtually global access point into account, and the development and maintenance of distributed services is independent of the life-cycles, schemas and availability of other parties.

Space and Data Models

The initial ideas for Triple Space were presented in [Fen04]. Tuples were defined to have the dimensions and semantics of RDF (⟨subject predicate object⟩) and are interlinked by use of URIs (like resource on the Web) to form graphs.

The tsc++ space is divided into semantic spaces that offer virtual data containers. The tsc++ space model defines a forest of disjoint space trees in a meronymic structure: any virtual space may contain multiple subspaces, while it can be contained in one parent. This allows layering and limited nesting of spaces in order to create a more expressive interaction platform. Each space is identified by a unique identifier (URL), independently of its tree affiliation, and delivers its own communication channel. Whenever a new interaction context, or a new group of agents is in consideration, a new virtual space is installed that links together the involved applications and data stores. This approach provides at least local scalability by grouping related data and users, and hence inherently limiting the scope and size of the interaction patterns [KSF07].
Interaction API

The operations of \texttt{tsc++} (Table 1) are based on recent complementary semantic space projects such as TripCom.\footnote{TripCom (IST-4-027324-STP, www.tripcom.org)} The API provides primitives for writing and reading sets of triples, seen as RDF graphs. Removal is currently not provided, as in distributed settings deletion of data, or even worse of knowledge - that includes inferred statements - inevitably leads to scalability problems. Omitting is might seem to contradict traditional database properties, however we argue that it can modeled via the available operations. Instead of removing data from a space, triples can be marked as invalid, or faded as proposed in [NTWM07]. Publishing a fading statement makes a triple less likely to be retrieved in a subsequent query, and eventually the fact disappears.

\begin{table}[h]
\centering
\begin{tabular}{|l|}
\hline
1 \texttt{write (Set(Triple) ts, URI s):URI} \\
2 \texttt{read (Template t, URI s):Set(Triple)} \\
3 \texttt{read (URI g, URI s):Set(Triple)} \\
4 \texttt{query (Template t, URI s):Set(Triple)} \\
\hline
\end{tabular}
\caption{Coordination primitives}
\end{table}

The ‘read’ returns an RDF graph, which are the triples that were published via the same write operation. In contrast, ‘query’ exposes a SPARQL CONSTRUCT behavior and returns all matching triples in a space, no matter which graph they belong to. In order to allow Web-like access to data, the traditional template-based ‘read’ is enhanced with a URI-based primitive (Table 1(3)) that extracts a graph by identifier. This primitive emphasizes the sought convergence of space and Web technology: the data (triples) is shared in a space, but retrieved as resources by their unique name, and not only by associative matching.

The semantic templates take the form of graph patterns; \texttt{tsc++} however also supports SPARQL CONSTRUCT queries. Graph patterns provide the most generic construct to determine the data to retrieve, and can be embedded in almost any RDF query language. They thus do not restrict the retrieval process, but leave it up to the matching algorithms and the repository to determine the query language, and the expressivity to use.

\section{3 System Architecture}

The functionality of the semantic space is realized by a set of software components referred to as kernel that run on all participating nodes. The components of the kernel are the Interaction APIs, the Operation Component with Metadata Manager, and the Coordination and Data Access Components (Figure 1).
The Operation Component implements the coordination primitives of (Section 2). Furthermore, it manages the kernel internal processes, which includes the transformation of user requests into formats that are processable by the underlying storage and coordination components.

The persistency framework (Data Access Component) is linked to the Operation Component via a Data Access API. This interface for storing and retrieving semantic data decouples the data sources’ models from the kernel. Query rewriting is done by the Data Access Component and is invisible to the other components; e.g. at retrieval a template is embedded into an expression that can be interpreted by the query engine of the data store. This decoupling allows the integration of any persistency framework: RDF stores, relational databases, file systems, or Web services that store triples remotely.

The management of distributed spaces, in terms of network access and of communication matters with other kernels, is the responsibility of the Coordination Component: i) it serves as a proxy for the resolution of requests to a space that is not co-hosted at the local kernel, and ii) it implements the access methods, distribution and discovery algorithms and any other network management-related process.

The Metadata Manager’s job is the creation and processing of metadata: content metadata about published data, and structural metadata about spaces in terms of space hierarchies. This metadata enables reasoning about structures and behaviors, and allows for optimized management procedures.

4 Technical Realization

To the best of our knowledge, tsc++ is the first fully distributed and open realization of a semantic space middleware. Other proposals did either not

---

2 A space can be accessed via kernels that do not share it; thus the need for proxies.
3 The latest open source release is available at http://tsc.sti2.at.
consider distribution of data (Semantic Web Spaces [NSAT07], TripCom), rely on proprietary middleware components (TSC, [FKS+07]) or are not yet open source (Sedvice [OHZ08]). With tsc++ we deliver a realization that offers an open and fully distributed implementation without a priori restrictions on the applicability and reusability.

4.1 Prototype Implementation

As stated previously, the functionality of a kernel is mainly provided by the Coordination and Data Access components:

**Coordination Component**

The core of the Coordination Component is based on the JXTA framework [Wil02] that offers many important features: simplicity and ease of implementation, openess, distributedness. This matches our objective of an openly available, distributed and scalable semantic space implementation. JXTA provides a generic P2P platform with a standard set of XML-based services and protocols with standardized schemas. Peers are the basic building blocks, create virtual overlay networks and might be any type of network device with unique identifiable peer IDs. An interesting trait of JXTA are the peer groups in which nodes self-organize private parts of the global network; peers can at any time belong to multiple peer groups. tsc++ uses this concept to model spaces. Kernels that authorize the same space are in tsc++ equal to peers of the same peer group, and the data that is published to the space is managed within the group. On top of this structure, we implement various distribution and discovery algorithms that use and extend the available JXTA protocols. Currently, tsc++ writes locally and queries remotely by means of flood, publish-subscribe, and random walker-based discovery. The different algorithms show their strengths or weaknesses depending on the system settings (Section 4.2). Flooding is simple and robust, has high tolerance to failures, low latency, and assures that all addressees eventually receive the messages. Search efficiency and completeness guarantees are in trade-off with response time and message overhead. In cases where scalability is not the primarily concern, as in corporate spaces, ‘flooding’ delivers a reasonable approach; it is however no solution at Web scale. Random walker-based discovery decreases the message overhead by only evaluating a query on the path along a random set of peers. This increases in turn the latency and decreases the completeness expectations. The type of implementation is influenced by the application requirements and the relevant trade-offs.
Data Access Component

As shown in Figure 1 the repositories are considered to be external to the kernel, and the Data Access Components translates the read templates into queries that can be interpreted by the store’s query engine. Currently, tsc++ supports the Sesame RDF store (Sesame 2.x) [BKvH02], OWLIM [KOM05], and to some extent YARS [HD05b]. Although these repositories use different query languages (SeRQL, N3QL, SPARQL), we can, by wrapping the templates according to the language, integrate them without altering the applications, API or other kernel components. This highlights the advantage of having a priori (only) graph patterns for our templates. For spaces whose stores all support SPARQL the flexible template interface allows however to expose the expressiveness of SPARQL also at the level of the API, and a kernel virtually provides a SPARQL endpoint on top of tsc++.

4.2 Evaluation

As to our knowledge, there are no comparable implementations, we cannot evaluate with a comparative analysis. Then, measuring the performance of a JXTA-based system is difficult, as there is no generic performance model available, and results largely depend on implementation details of the application and the JXTA reference implementation [HD05a]. We thus concentrate on scalability and performance indicators with respect to varying numbers of spaces, kernels and triples.

As tsc++ is based on local write only, the performance of ‘write’ is equivalent to the performance of the repository: our experiments show an almost unmeasurable 20ms as upper limit for writing triples. We therefore limit the evaluation to the retrieval operations and depict the results of ‘read’ by means of two performance measures:

1. The number of messages per request: due to the nature of the flooding approach, we expect a linear base component with respect to the number of kernels that share a space.

2. A latency indicator: the time depends on the infrastructure, and is thus to consider with care; however, it showcases the relative change in latency.

The test results are shown in Table 2, and compare the results of the flooding and walker-based algorithms. The experiments were conducted with 1, 10 or 20 spaces that are shared by 1, 10 or 20 kernels. The evaluation setting thus defines \( n \) kernels that store data, and one kernel per scenario that reads from the \( n \). Retrieval was configured to not consider local data. In this way, we focus on the performance in distributed environments.
Table 2: Evaluation Results for Read

<table>
<thead>
<tr>
<th>Spaces</th>
<th>1</th>
<th>10</th>
<th>20</th>
<th>1</th>
<th>10</th>
<th>20</th>
<th>1</th>
<th>10</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flooding</td>
<td>Msg/read</td>
<td>2</td>
<td>11</td>
<td>21.1</td>
<td>2</td>
<td>11</td>
<td>21.1</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>time [s]</td>
<td>0.25</td>
<td>0.71</td>
<td>1.31</td>
<td>0.23</td>
<td>0.69</td>
<td>1.47</td>
<td>0.21</td>
<td>0.67</td>
<td>1.24</td>
</tr>
<tr>
<td>Walker</td>
<td>Msg/read</td>
<td>2</td>
<td>6.7</td>
<td>9.5</td>
<td>2</td>
<td>6.4</td>
<td>11.4</td>
<td>2</td>
<td>6.4</td>
</tr>
<tr>
<td>time [s]</td>
<td>0.27</td>
<td>0.73</td>
<td>1.54</td>
<td>0.28</td>
<td>0.89</td>
<td>1.47</td>
<td>0.29</td>
<td>0.77</td>
<td>1.98</td>
</tr>
</tbody>
</table>

The setup consisted of a server hosting up to twenty virtual kernels; each launched in its own execution environment. One kernel instance consumed up to 20 MB of runtime memory. The published graphs contained between ten and twenty triples and were equally distributed over the test kernels. The number of triples was kept low, in order to focus on the discovery and not the transmission and processing time. Retrieval was governed by a simple triple pattern of the form <S ?p ?o>, where the subject ‘S’ was randomly chosen out of the possible subjects within the sample graphs. A response messages had then an average size of about 2kB.

A first observation shows that the number of spaces has neither an influence on the message count, nor on the retrieval time. This is explained with the fact that spaces are virtual entities at the level of the repositories only. While joining a space is time-wise a rather expensive operation (16-20 seconds), not using a proxy pays of once a node must manage increasing numbers of interactions. In fact, a proxy kernel has to temporarily join a JXTA peer group in order to interact with the space. This process is as costly as joining a space, and retrieval operation suddenly require noticeable more time.

Flood-based discovery has, as theory suggests, a message count that increases linearly with the number of spaces. For the trials with twenty nodes, some of the fifty test graphs were stored multiple times, which resulted in requests that were answered by two kernels. Therefore the total number of messages slightly exceed the expected 21 (20 flooding messages plus at least 1 response). Furthermore, we observe that the latency increases with the number of kernels. This might be, because all our test kernels had to be run on one server machine, or because in JXTA, some peers provide centralized management of the peer groups and become bottlenecks. Still, we argue that the latency shows quasi-constant behavior, or at worst a linear increase. This is acceptable, in particular as the tested setting chooses to query the whole network and hence to seek completeness for the price of latency. For the walker-based proposal, the number of messages shows still linear behavior, although roughly cut in half compared to flooding.

4 Although we worked only with virtual distribution, we guarantee the described functionality also across multiple networks.
The implemented walker inherits flooding-style search, with the difference that messages are only passed on if a questioned kernel does not host the required data. As in our experiments, the RDF graphs are randomly stored on any of the kernels, the answers are discovered (in average) at kernel $n/2$, and message count is still in $O(n)$. The latency increases however faster with increasing numbers of kernels, also still linearly. For spaces that are shared by a large number of kernels with a significant request load, this alternative will eventually pay off; further improvements are however indispensable, e.g. by exploiting knowledge about data and queries for more selective walkers.

5 Related Work

Based on the initial ideas for combining tuplespaces with the Semantic Web [Fen04], several systems have been designed and to some extend been implemented. Within the project TSC, a space-based middleware for Semantic Web services has been developed. The TSC implementation is built upon an existing proprietary coordination middleware which led to many design decisions being simply carried over rather than being re-assessed. This and the fact that there are further scalability problems, lead to the development of tsc++, which – also due to its alignment with recent results from the TripCom project – is clearly compatible with both space-based computing and the Semantic Web.

We also point to the work on "dataspaces" by [FHM05]. They manifest the need for loosely coupled solutions to storage integration. Their framework does not rely on design time semantic integration, but rather on the co-existence of data sources. Dataspaces deliver a layer of core functionality on top of the data providers and virtually expose a data integration platform. This allows applications to focus on their own functionality rather than on data integrity and efficiency of integration. Furthermore, keeping individual sources on different machines preserves the advantages of distributed systems: decentralization, robustness, and scalability. Thus, tsc++ can be seen as realization of such concepts from a Semantic Web perspective, which inherently adds further means for integration: data mediation and reasoning.

6 Conclusions

The paper revisited Triple Space middleware with a focus on architecture and implementation in large scale, open and distributed Web environments. We presented tsc++, an implementation of the Triple Space paradigm on top of an existing P2P framework. tsc++ materializes many of the benefits that are expected from semantic middleware and that are necessary to realize large scale applications over distributed semantic artifacts. A well-received example was
established by the TripCom project with the eHealth showcase. A distributed Triple Space platform is used to realize a European Patient Summary infrastructure that enables the sharing of and easy access to medical records, independently of political and institutional boundaries, heterogeneities in technology and data or patient security and data protection policies [CKSF09, KSC+09].

The evaluation of tsc++ shows that many challenges remain in implementing scalable semantic spaces. The plans are to examine how approaches to scalability in space-based systems may be applied to semantic spaces and the Semantic Web, and to further implement and test our approach. Alternative distribution and discovery algorithms have to be investigated and integrated, while complementary implementations that are not based on P2P or JXTA should be considered too; e.g. semantic spaces over Grid. Ongoing work is moreover concerned with exploiting the available metadata and enabling adaptivity through scalability-driven trade-offs.

While there is still work to do in terms of establishing the principles and technologies at large scale, we argue that tsc++ is an important step towards a global infrastructure into which arbitrary information providers (e.g. databases, Web services) and consumers (applications) can be plugged at will. In that way, the Semantic Web can become one big virtual data store that provides knowledge anywhere, anytime to any application.

References


