Journal of Universal Computer Science, vol. 18, no. 2 (2012), 286-311
submitted: 23/3/11, accepted: 16/1/12, appeared: 28/1/12

RESTifying a Legacy Semantic Search System: Experience and Lessons Learned

Guillermo Vega-Gorgojo, Eduardo Gómez-Sánchez
Miguel L. Bote-Lorenzo, Juan I. Asensio-Pérez
(School of Telecommunications Engineering, University of Valladolid
Paseo de Belén 15, 47011 Valladolid, Spain
{guiveg, edugom, migbot, juaase}@tel.uva.es)

Abstract: The REST architectural style pursues scalability and decoupling of application components on target architectures, as opposed to the focus on distribution transparency of RPC-based middleware infrastructures. Ongoing debate between REST and RPC proponents evidences the need of comparisons of both approaches, as well as case studies showing the implications in the development of RESTful applications. With this aim, this paper presents a revamped RESTful version of a legacy RPC-based search system of educational tools named Ontoolsearch. The former version suffers from reduced interoperability with third-party clients, limited visibility of interactions and has some scalability issues due to the use of an RPC-based middleware. These limitations are addressed in the RESTful application as a result of applying REST constraints and using the Atom data format. Further, a benchmarking experiment showed that scalability of the RESTful prototype is superior, measuring a $\sim$3 times increase of peak throughput. In addition, some lessons learned on RESTful design and implementation have been derived from this work that may be of interest for future developments.

Key Words: RESTful web services, remote procedure call, semantic search, educational tools
Category: C.2.4, H.3.3, K.3.1

1 Introduction

Over the past few years, the Web has evolved into a computing platform with the advent of websites providing search, shopping, auction or encyclopedia services to name a few [Raman, 2009]. These services rely on Representational State Transfer (REST) [Fielding, 2000], the architectural style underlying the Web. REST defines a set of constraints that induce some desirable properties on target architectures. More specifically, REST attempts to minimize latency and network communication, while at the same time maximizing the independence and scalability of application components [Fielding and Taylor, 2002]. As a result, REST enables the development of highly flexible, decoupled and scalable distributed applications as exemplified by the Web itself.

Conversely, much of the work in the distributed systems research field has focused on hiding the complexity of distribution as much as possible from application programmers [Coulouris et al., 2005, ch. 1]. This hiding is referred to as transparency and is typically achieved through the use of a middleware infrastructure [Bernstein, 1996]. Some remarkable middleware examples are the Common Object Request Broker Architecture (CORBA) [OMG, 1991], Enterprise Java Beans [Sun, 1999] and Web Services [Curbera et al., 2002]; all of them are based on the remote procedure call (RPC) abstraction which aims to make local and remote invocations indistinguishable from the programmer's point of view. Since the middleware infrastructure hides all the network communication, it allows developers to write exactly the same code for either type of call.

Despite programmer convenience of RPC-based middleware, [Waldo et al., 1994] and others convey that such attempts lead to the development of systems that are fragile, prone to errors and restricted in scale due to the differences between local and distributed computing. Remarkably, [Vinoski, 2008a] and [Richardson and Ruby, 2007] claim that these limitations can be addressed applying REST constraints. While the debate between RPC and REST proponents remains vivid [Pautasso et al., 2008], specially in the blogosphere, the number of RESTful web services follows a steady growth led by major Web players such as Google or Amazon.

Although REST claims seem to be accepted nowadays, there are some indications that the industry have not yet fully adopted – or even understood – REST principles. For instance, [Maleshkova et al., 2010] presents a recent survey of Web APIs on the Web, finding that only one third can be considered RESTful, almost half of the Web APIs are RPC-based, while the remaining 20% are considered hybrid – i.e. RESTful in purpose, but failing to comply with HTTP semantics. Furthermore, [Pautasso and Wilde, 2011, p. 5] warns that the current landscape of REST is still in development, so more research efforts are required to facilitate the design and implementation of truly RESTful systems, as well as to test their design qualities in a systematic way. As a result, there is a need to assess in practice the intended benefits of REST, as well as to provide more precise guidelines for the development of RESTful web services. With this aim, the authors present here a revamped RESTful web version of an existent search system of educational tools named Ontoolsearch [Vega-Gorgojo et al., 2010]. This system was conceived to allow teachers to discover tools for supporting their learning scenarios. Note that search systems like Ontoolsearch are specially appealing with the increasing use of technology in learning and teaching activities [Richardson, 2010], specially with the emergence of the Web 2.0 movement [O'Reilly, 2005] and the proliferation of Web-based applications.

The original Ontoolsearch system is implemented as a client-server application using the well-known Java Remote Method Invocation (Java RMI) [Sun, 2004], an RPC-based object-oriented middleware. Although this prototype of Ontoolsearch has proved to be reliable for searching educational tools, there are also some drawbacks due to the use of an RPC-based middleware. In particular, the custom Java RMI server interface severely restricts the use of third-party clients since these should be specifically designed for this purpose. Moreover, there are some scalability issues of this version, thus limiting future plans of using Ontoolsearch at a larger scale. In this regard, the REST architectural style was considered a suitable alternative since it promotes both the interoperability and scalability of application components [Fielding and Taylor, 2002]. Therefore, this proposal can serve as an insightful case study to illustrate the transition from a legacy application to a RESTful one, highlighting the technical merits and the encountered limitations. It should be noted that developing a REST system entails the use of different abstractions and assumptions from traditional RPC-style systems; consequently, a major contribution of this work is to provide some guidelines and recommendations for REST design and development based on the authors' experience of refactoring a legacy RPC-based search system, presented here as a case study. In addition, intended benefits of REST are assessed in a feature evaluation and a benchmarking study by comparing the RESTful version and the legacy RPC-based search system. Note that research works aiming to assess REST benefits tackle this issue at a rather abstract level, e.g. [Pautasso et al., 2008] and [Adamczyk et al., 2011]; in contrast, this work is novel in the sense that REST benefits are assessed empirically in the context of the presented case study.

The rest of this paper is organized as follows: section 2 briefly describes the REST architectural style and the former version of Ontoolsearch, outlining its limitations. Section 3 presents the new RESTful version of Ontoolsearch, including some discussions and reflections about REST design and implementation. Next, both versions are compared in a feature analysis study and a benchmarking experiment in section 4. Then, section 5 draws some lessons learned derived from this study. Finally, the main conclusions of the study are shown as well as current research work.

2 Background

This section begins with an overview of the REST architectural style. Then, the former version of the Ontoolsearch system is succinctly described, as well as the limitations that have been identified.

2.1 The Representational State Transfer architectural style

Representational State Transfer (REST) [Fielding, 2000] is a software architectural style consisting of components, connectors and data elements constrained in their relationships in order to achieve a desired set of properties. Components are processing elements that provide a transformation of data, while connectors mediate communication among components by transferring data elements without changes. In REST the main data element is the resource, the intended conceptual target (e.g. a book) of a request, which has an addressable identifier (e.g. a URL) and whose state can be exchanged among components through representations (e.g. a PDF document).

The set of constraints defined in REST are: client-server architectural style, stateless communication, cacheable, uniform interface, layered system and optionally code on demand (see [Fielding, 2000, ch. 5] for a thorough discussion). The rationale of these constraints is to minimize latency and network communication, while at the same time achieving scalability, independent deployment of components and visibility of interactions. The key constraint for achieving those goals is the uniform interface that is enforced to be the same in all component interfaces. As a result of applying the principle of generality, the overall architecture is simplified and more decoupled, thus allowing clients and servers to evolve independently.

A realization of the REST architectural style is the World Wide Web, demonstrating the validity of REST pursued benefits for an Internet-scale, multi-organization and anarchically scalable information system. In this regard, the core architectural standards for the Web are Uniform Resource Identifier (URI) [Berners-Lee et al., 2005] and Hypertext Transfer Protocol (HTTP) [Fielding et al., 1999]: a URI is a sequence of characters that univocally identifies a resource, while HTTP is a stateless application protocol designed specifically for the transfer of resource representations in the Web. Noteworthy, HTTP exemplifies the uniform interface constraint, defining a balanced set of methods – primarily `GET`, `PUT`, `POST` and `DELETE` – for accessing resources.

With the success of the Web, a new breed of applications have appeared using the Web as platform and following REST principles [Raman, 2009]; sometimes called RESTful web services [Richardson and Ruby, 2007], examples include mapping services, e.g. Google Maps[1], blog publishing services, e.g. WordPress[2], or online storage services, e.g. Amazon S3[3].

2.2 Former version of Ontoolsearch

Ontoolsearch [Vega-Gorgojo et al., 2010] is a search system of educational tools specifically designed for teachers. In stark contrast to conventional search facilities, a novel feature of Ontoolsearch is the formulation of semantic searches [Guha et al., 2003] for requesting tools, using the concepts defined in the Ontoolcole ontology [Vega-Gorgojo et al., 2008]. This way, it is possible to look for tools with a specific functionality, either referring to a particular tool type, e.g. *a*

[1] `http://maps.google.com/`
[2] `http://wordpress.org/`
[3] `http://aws.amazon.com/s3/`

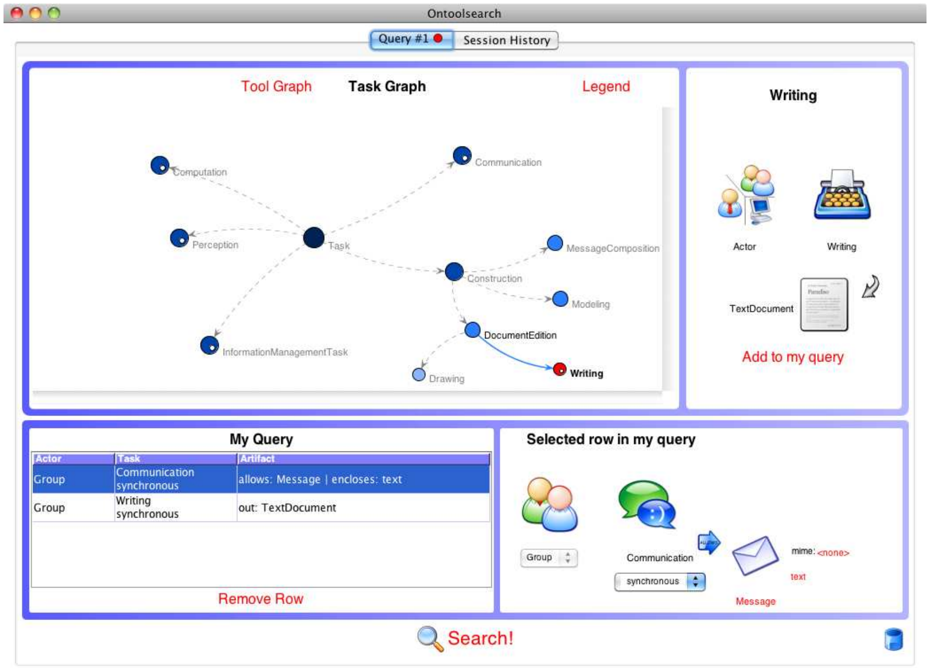

Figure 1: Snapshot of the search user interface of Ontoolsearch during the formulation of a query in which the user wants to find tools that can be employed by groups to synchronously write a document and exchange text messages.

concept map tool, or specifying a set of tasks that should be supported, e.g. *any tool that allows a group to synchronously communicate exchanging text messages*.

In order to meet teachers' needs, the user interface was devised following a participatory design strategy [Muller and Kuhn, 1993]. The agreed design offers an innovative graphical direct manipulation interface, enabling the formulation of semantic searches referred to the concepts defined in Ontoolcole (using tool and task graphs for query formulation). With this approach, the aim is to simplify the process of query construction, hiding the formalism and the implementation details of the underlying ontology to teachers. Moreover, the representation of visual elements in the user interface should allow the rapid communication of the abstractions modeled in Ontoolcole, since humans are highly attuned to images and visual information [Larkin and Simon, 1987]. Figure 1 shows a snapshot of the search user interface during a query formulation process.

A prototype of Ontoolsearch was implemented in Java, following the well-known client-server architectural style. The client component includes all the user interface functionality for constructing queries and showing the results; in addition, the client communicates with the server component to request tools compliant with submitted queries. The server handles incoming requests extracting the query element, computes the results and sends them back to the client component. The actual query processing is performed by a Description Logics [Baader et al., 2003] reasoner attached with the tool dataset. Required client-server communication is supported with the Java RMI object-oriented middleware, encapsulating queries as Java objects and using HTTP tunneling to bypass firewalls.

To evaluate this system, a group of 18 teachers was engaged in a formal comparison study of Ontoolsearch with a keyword search facility based on the well-known text search engine Lucene[4]. The main goal of the study was to assess whether semantic searches with Ontoolsearch could be preferable to conventional keyword-based searches in this context. Noteworthy, retrieval performance was 31% better with Ontoolsearch in spite of participants' previous experience with keyword searches. Indeed, typical problems of keyword searches were detected (specially synonymity), while teachers succeeded in formulating semantic searches with Ontoolsearch. In this sense, they pointed out that graph browsing facilitates the discovery of unknown tools and allows them to easily assess tool functionalities. In contrast, this capability is difficult to achieve with keywords since a search facility does not give clues about the constitution of the tool database. Overall, these results suggest teachers are likely to adopt Ontoolsearch for conducting tool searches (see [Vega-Gorgojo et al., 2010] for more details of this study).

Beyond this evaluation, the Ontoolsearch system was made freely available to the teacher community in November 2008, being sparsely employed to search tools for educational settings. During this time, users have requested some new features, namely, support keyword-based queries and tool browsing, that are included in the RESTful version (see section 3).

2.3 Limitations of the former version of Ontoolsearch

Although the Ontoolsearch system has proved to be reliable during its operation time, there are also some limitations that should be tackled:

1. Limited interoperability with third-party clients. RPC-based middlewares are grounded on the definition of service interfaces composed of a set of operations and their parameters. However, resulting interfaces inhibit general reuse since only purpose-built clients can invoke them [Vinoski, 2008b]. For this reason the authors offer a specific Ontoolsearch client to access the service functionality. Nevertheless, Ontoolsearch client and server are tightly

[4] `http://lucene.apache.org/`

coupled to a Java RMI interface, so a small change would require the replacement of deployed clients. Moreover, some teachers could prefer the use of an existing client application for posing queries or visualizing the results, instead of the provided Ontoolsearch client.

2. Performance and scalability issues. Since Ontoolsearch requires to operate in an Internet environment, HTTP tunneling is employed to communicate through firewalls and proxies. However, tunneling in Java RMI has a profound impact on performance; [Juric et al., 2004] reports about an order of magnitude worse. With respect to scalability, the fact that communication is stateful in Java RMI poses a bottleneck for scalability since the server has to manage session state. While a performance boost is always appreciated by users, scalability gains are of great importance to accommodate increases of the workload in the mid-term. In this regard, future plans for Ontoolsearch include its deployment in Virtual Learning Environments for supporting the discovery and integration of third-party tools [Alario-Hoyos and Wilson, 2010], so scalability is a desired architectural property to achieve.

3. Limited visibility of interactions. Keeping session state in the server makes specially difficult to assess the nature of a request through the inspection of interactions in Java RMI. Therefore, mediation, e.g. via a shared cache, and monitoring of interactions in a communication exchange are extremely difficult to achieve [Vinoski, 2008a]. Moreover, limited visibility of interactions precludes the creation of tool or search bookmarks in Ontoolsearch.

Since all these limitations are due to the use of an RPC-based middleware, an architectural change was considered. Remarkably, some of the intended benefits of REST apply to these limitations: (1) independent evolvability and enhanced interoperability by means of REST uniform interface, (2) improved performance and scalability through caching and stateless communication, and (3) visibility of interactions, again due to the uniform interface and stateless communication constraints. Therefore, all these limitations could be addressed by RESTifying the Ontoolsearch system, which implied an architectural redesign and the development of a new prototype.

3 A RESTful version of the Ontoolsearch system

In this section, the design of the RESTful version of Ontoolsearch is presented. It follows a discussion of REST design, stressing its principles and implications. Then, the new prototype of Ontoolsearch is presented, illustrating the transition from a resource design to a RESTful system. This section finishes with some reflections about the implementation of RESTful applications.

3.1 RESTful design of Ontoolsearch

The new version of Ontoolsearch has been designed to support the following functionalities:

F1 Semantic search. As in the former version, the client component is in charge of composing semantic queries, submitting them to the server component, retrieving the results and performing the rendering. Correspondingly, the server extracts the query for each request, computes the results and sends them back to the client.

F2 Semantic search browsing. Teachers should be able to gather their submitted searches and associated results, as well as allowing to share obtained results with other users of the system. Therefore, the server has to maintain a record of submitted searches that should be browsable. Note, however, that clients are not allowed to delete previous searches.

F3 Keyword search. A set of keywords can be submitted to the server which obtains the tool descriptions that match the query terms and sends the results back. In this case support for browsing is not required, so the server does not have to keep previous keyword searches.

F4 Tool browsing. Besides searching, browsing the tool dataset should be supported, allowing clients to browse the members of a tool category or the full list.

Note that former version of Ontoolsearch only supported functionality F1, while the remaining three are new. Moving to the design of the RESTful version, the procedure defined in [Richardson and Ruby, 2007, ch. 4–6] has been followed; it consists of a series of steps marked in bold face in the subsequent paragraphs, together with the discussion of the design decisions that they involved in the case of Ontoolsearch.

In the design process, **Step 1 is figuring out the data set** and **Step 2 is splitting it into resources**. According to this, the data model includes semantic queries, keyword queries, tools, tool types and corresponding collections. The resulting arrangement of resources can be shown pictorially in Figure 2, grouping resources by their functionality. This same diagram illustrates **Step 3, naming the resources with URIs** and **Step 4, exposing a subset of the uniform interface** of the followed procedure; more details are given next for each resource.

The `Root` resource is the main entry point of the application, returning a welcome HTML page to web browsers accessing the service through a `GET` request. Note that the chosen URI of this resource is `http://www.gsic.uva.`

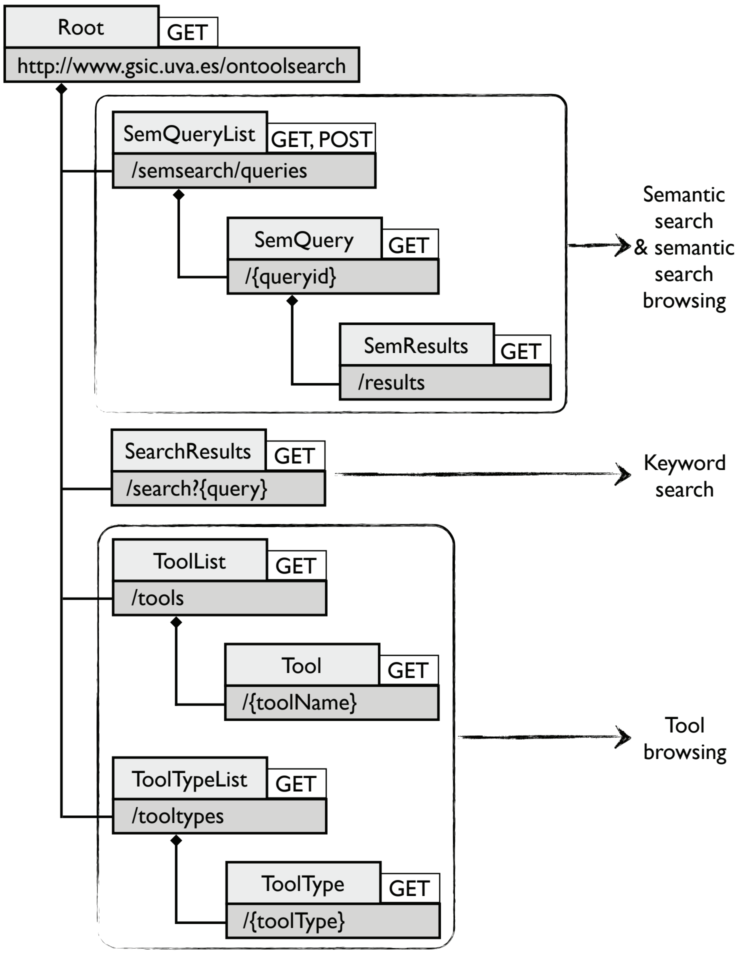

Figure 2: Resource design of the RESTful version of Ontoolsearch, including the URI design and exposed HTTP methods for each resource type.

`es/ontoolsearch` and it contains hyperlinks to the core functionalities of Ontoolsearch: semantic search, semantic search browsing, keyword search and tool browsing.

With respect to semantic search and semantic search browsing, the entry point is the `SemQueryList` resource that represents the collection of the semantic queries submitted to the system. This collection can be gathered by means of a `GET` request, thus allowing users to browse the semantic queries and their results in compliance with functionality F2. The creation of a new member of the collection is handled by this resource through a `POST` request encapsulating the submitted query. Success of this request triggers the creation of a `SemQuery` resource as well as its subordinate `SemResults` with the results of the submitted query. Again, the actual value of a semantic query or its results can be obtained through a `GET` request to the target resource.

For handling keyword-based queries, `SearchResults` is the unique resource employed, corresponding to a list of search results. Note that the query is an integral part of the URI of `SearchResults` (see Figure 2), since the conceptual target of this resource is the set of tools that match the search criteria. Such encapsulation of queries in URIs is an established practice in RESTful web services[5] [Richardson and Ruby, 2007, pp. 121–123]; however, heed that the approach taken in the precedent case is different, modeling semantic searches as `SemQuery` resources. The rationale of this decision is to support semantic search browsing, while keyword searches are not recorded (see the functional requirements above).

Concerning the functionality of tool browsing, it is achieved exposing the resources `ToolList`, `Tool`, `ToolTypeList` and `ToolType`. `ToolList` corresponds to the list of all the tools available in the dataset and each member of this set is a `Tool`. Similarly, tool types, e.g. `ConceptMapTool` or `VideoconferenceTool`, can be browsed by means of the `ToolTypeList` resource whose members correspond to `ToolType`. Note that only the `GET` method is available for retrieving the value of these resources, since users are not allowed to create new tools or tool types (thus requiring `POST` or `PUT` requests).

At this stage the set of resources exposed by the RESTful version of Ontoolsearch is defined, **Step 5 is deciding the representations that should be accepted and served**. Adhering to an existent and proven representation format is specially important since it can reduce the design effort and, at the same time, it facilitates the use of the application by third-party clients. In this regard, one of the formats considered was the Atom Syndication Format [Nottingham and Sayre, 2005], an IETF standard that defines an XML vocabulary for describing lists of related information – known as *feeds* – composed of a number of items which are called *entries*. Atom defines an extensible set of elements for describing associated metadata of feeds and entries, e.g. `title`. As a measure of the popularity of Atom, this representation format is extensively employed in the Web, specially in blogs and news sites, and, more importantly, there are many available Atom clients including Google Reader[6] and browsers such as Firefox[7] and Safari[8].

Given the extensive use of lists in Ontoolsearch – specifically, collections of semantic searches, query results, tool types and tools are employed – Atom seems

[5] For example: `http://www.google.com/search?q=Ontoolsearch`
[6] `http://www.google.com/reader/`
[7] `http://www.firefox.com/`
[8] `http://www.apple.com/safari/`

a suitable representation format. Since Atom is conceived for syndication, it is possible to subscribe to a feed with the performed queries, which may be of special interest to a system administrator. There are other cases of Atom syndication that should be useful to teachers; for instance, they can subscribe to the tool feed `http://www.gsic.uva.es/ontoolsearch/tools` or to a specific tool type feed, in order to discover new tool entries incorporated to the tool dataset. Moreover, associated metadata of Ontoolsearch resources can be easily mapped to Atom elements such as `author`, `title` or `summary`; indeed, the `category` element is specially convenient to include tool type information. To illustrate the mapping of Atom to Ontoolsearch resources, Figure 3 shows the visualization in Firefox of the `BulletinBoard` feed that includes some instances of that tool type as entries[9]. Likewise, semantic searches can be codified in Atom, using the `content` element of an entry to include the actual semantic query. Therefore, the benefits of using Atom can be summarized as follows: 1) improved interoperation with third-party clients, thus enabling the reuse of existing Atom clients; 2) facilitate the discovery of new resources to users through web syndication, such as new tools to teachers and submitted queries to system administrators; and 3) avoid the effort to develop a customary representation by adhering to an existent, proven and widely-employed representation format such as Atom.

To conclude this subsection, some remarks are given about the remaining steps in the followed procedure.

Step 6, modeling resource relations with hyperlinks, refers to the "hypermedia as the engine of application state" principle [Fielding, 2000, ch. 5], i.e., resource representations should include hypermedia controls (e.g. links) to drive the transition to new application states, thus freeing clients from having to know about the resource URIs of a RESTful service [Parastatidis et al., 2010]. In this regard, the relationships shown in Figure 2 are modeled using the capabilities of Atom. For instance, every feed includes an entry to the `Root` resource (see last item in Figure 3). Another example of connectedness is the use of the `link` element to point to other related resources, e.g. every tool entry has (a) link(s) to its tool type(s). Moreover, both keyword and semantic search results feeds contain an entry for each tool obtained. Finally, **Step 7 is to consider typical course of events and error conditions**. Thus, the flow of interactions has been analyzed for each functionality of Ontoolsearch, checking the HTTP requests and responses needed to fulfill a petition. Further, possible errors, e.g. submitting a semantic search in an unsupported format, have been considered including the HTTP status codes [Fielding et al., 1999] to be returned.

[9] Note, however, that Firefox does not render all the information included in an Atom feed; in this particular case metadata of authors, categories and hyperlinks typed as `alternate` are not shown.

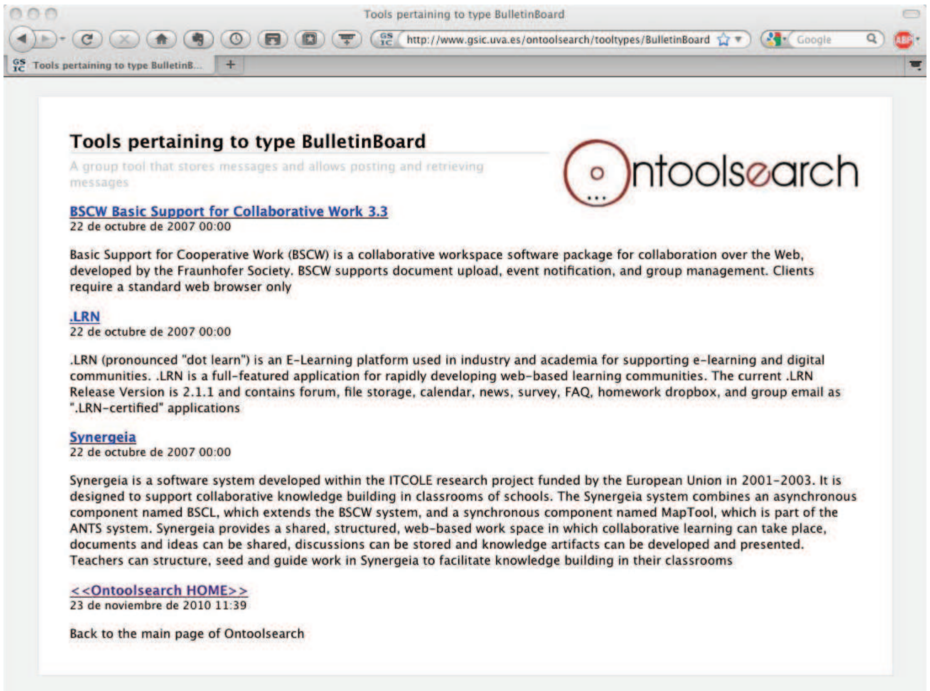

Figure 3: Visualization of the `BulletinBoard` tool type Atom feed with Firefox.

3.2 RESTful design discussion

Designing a RESTful web application is very different to an RPC-style application. In the latter case, the focus concentrates on the definition of the service interface, composed of a set of operations and their parameters. In this way, a remote call looks no different than a local one to programmers, since the RPC middleware aims to hide the complexity of distribution as much as possible. In contrast, the fact that the application is distributed becomes evident in RESTful design, so a different perspective is required. Therefore, the designer has to deal with the resource abstraction, since resources are the "visible" elements of a RESTful application. Moreover, resources are accessed through HTTP methods (an exemplification of the uniform interface constraint), so an understanding of HTTP semantics is required.

It should be taken into account that resource design must be done with care; resources should represent the objects of interest of the application. Depending on the domain they can differ in many ways – see for example the resource design of Ontoolsearch in Figure 2. However, it is important not to expose actions as resources [Richardson and Ruby, 2007, p. 117]: an operation of unknown purpose will be made when called, thus breaking the semantics of the employed HTTP method and the uniform interface constraint. Note that in some cases the result of exposing an action as a resource would be accidentally RESTful as in the following example: `/doSearch?q=whiteboard` is a valid URI that can be bookmarked and doing a `GET` on this resource complies with the semantics of this HTTP verb. Unfortunately, adding other "non-read" functionalities would break this illusion, e.g. trying to delete a semantic search through a `GET /semsearch/queries/123/delete` request. Due to this, resources should always represent data elements – whichever the application domain is – and never operations. A related pitfall is the so-called *overloaded POST* [Richardson and Ruby, 2007, p. 101], i.e. sending `POST` requests to perform any processing, while the actual method information may be in the URI, the HTTP headers or the entity-body. As in the precedent case, the overall effect is to break the semantics of the uniform interface, so the use of overloaded POST is discouraged.

Another important decision to be taken is the representation formats to be accepted and served. Choosing an existing format can avoid the effort of designing a purpose-built one and may broaden the audience of the service. For these reasons creation of *ad hoc* representations such as XML documents is discouraged, while available formats for the application domain should be checked, e.g. microformats[10]. Note that the decision of employing the Atom format in Ontoolsearch enables the use of an extensive list of third-party clients to access this system, in contrast to the former Java RMI version. In addition to this, a nice feature of HTTP to reduce data coupling is content negotiation [Fielding et al., 1999] which allows serving alternative representations of a resource to clients depending on their preferences. Though Atom is the only format supported in Ontoolsearch, a new one could be easily incorporated in the future without impact on Atom clients, e.g. a JSON format for use in an eventual AJAX client running on a browser.

A final remark is made about the hypermedia principle of REST. Resource representations should contain hyperlinks to other resources in order to provide feedback to clients about available transitions to new application states. For instance, a semantic query feed in Ontoolsearch contains an entry that points to the results of this query, thus allowing clients to follow this link in order to present retrieved tool instances. Note that the hypermedia principle reduces client-server coupling, since clients do not have to know beforehand the resource URIs exposed by a service. Accordingly, the flow of events of a RESTful application should be carefully designed through hyperlinks to avoid undesired application state transitions.

[10] `http://microformats.org/`

3.3 RESTful implementation of Ontoolsearch

So far, the resource design of the RESTful version of Ontoolsearch has been completed. Moving to the implementation phase, a RESTful web framework should be chosen in order to ease the development process as much as possible. Specifically, the Restlet Java framework[11] has been employed to develop the RESTful version of Ontoolsearch. Restlet aims to provide a lightweight and comprehensible web framework that sticks close to REST constraints. Another key goal of Restlet is to provide a unified framework for web applications, removing the distinction of client- and server-side APIs. This way, it is possible to create a single component in Restlet that acts both as a client and as a server. Moreover, Restlet includes extensions for the Atom representation format and the Apache Solr[12] text search system that have been employed in the development of the prototype.

Developing a RESTful application with Restlet is simplified by the fact that the terminology employed matches the elements of REST, defining classes such as component, connector, resource or representation. To illustrate this, Figure 4 shows the resulting physical architecture of the RESTful web version of Ontoolsearch using Restlet. At the core there is a `Component` that hosts the `Ontoolsearch Application`, a coordinated set of objects that provide the desired functionalities. This component has attached an `HTTP Server Connector` that transfers all requests to the application. Then, the `Router` is in charge of dispatching incoming calls to resources. This is done by specifying a set of routes that maps URI templates to target resources (see these routes in Figure 4).

The resources held by the application correspond to the resources designed in subsection 3.1; indeed, they are subclasses of the `ServerResource` Restlet class and implement the HTTP methods defined in Figure 2. Using the Atom extension, resources transfer feed representations back and forth on response to client requests, e.g. `ToolList` responds to a `GET /tools` call with the feed of available tools. Note, however, that resources rely on other application-specific objects to obtain their actual value. In this regard, the `ToolRegistry` loads the tool dataset available as a set of RDF dump files at `http://www.gsic.uva.es/ontologies/`, while `ToolList`, `Tool`, `ToolTypeList` and `ToolType` resources get their values from the `ToolRegistry`. Moreover, this registry has attached an off-the-self `OWL Reasoner` that is in charge of processing semantic queries as requested by `SemQueryList` when a `POST /semsearch/query` call arrives, thus resulting on the creation of a query and its results. Correspondingly, the collection of submitted semantic queries and obtained results are gathered by `SemQuery` and `SemResults` to handle incoming requests.

With respect to the keyword search functionality, a `Solr server` is set up

[11] `http://www.restlet.org/`
[12] `http://lucene.apache.org/solr/`

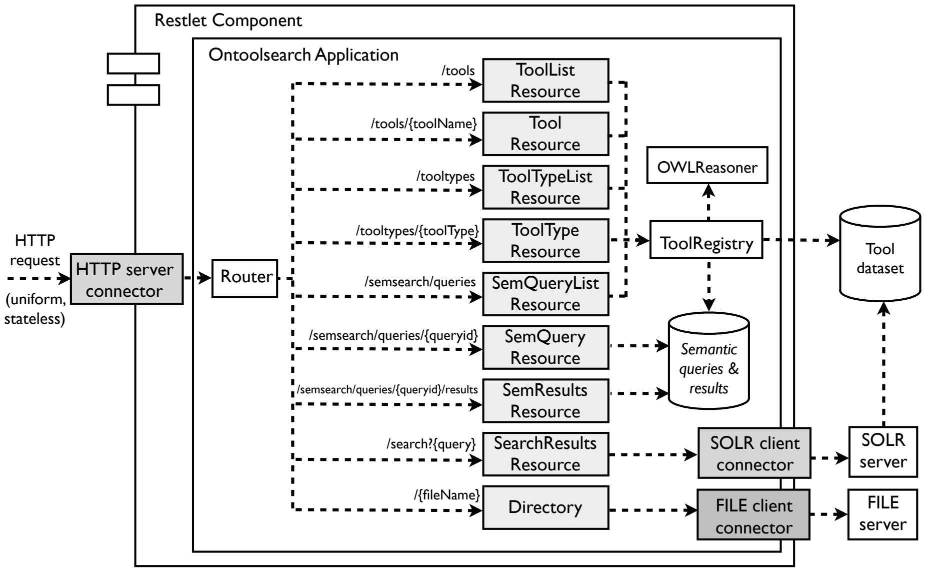

Figure 4: Physical architecture of the RESTful version of Ontoolsearch.

to index the tool dataset and to respond to keyword queries. Therefore, a request such as `GET /search?q=whiteboard` is dispatched to `SearchResults` resource which forwards the query to the Solr server through the `Solr client connector` provided with Restlet; obtained responses are then handled by `Search Results` returning a feed to the client. Finally, `Directory` is a Restlet class that serves static files. It has been configured to dispatch the `index.html` file and other resources (Ontoolsearch images and tutorials basically), thus fulfilling the role of the `Root` resource in Figure 2.

The new RESTful prototype of Ontoolsearch can be accessed at `http://www.gsic.uva.es/ontoolsearch/`. As regards the client side, existing third-party Atom readers can be employed without changes to browse Ontoolsearch exposed resources, thus greatly improving the potential audience of the system. In addition, the root `index.html` file includes a simple HTML form with a text box for submitting keyword queries to the system. However, posing a semantic query directly in an Atom feed is an arduous task since Atom serves as a mere envelope in this case. To overcome this limitation, former version of the Ontoolsearch client has been modified to communicate with the new RESTful version of the server. Noteworthy, changes did not involve much effort since the user interface was maintained (see Figure 1) and Restlet facilitates the development of clients that consume and manipulate resources by means of the `ClientResource` class.

3.4 RESTful implementation discussion

Implementing a RESTful system from scratch can be a daunting task, so the use of a framework may be considered. In this regard, there are REST frameworks in many languages, e.g. Ruby on Rails[13] for the Ruby language. In the particular case of Ontoolsearch, only Java-based frameworks were analyzed in order to reuse some code from the former version and to easily integrate existing semantic- and text-based search engines that are typically available in Java. Besides Restlet, Jersey[14] and Apache CXF[15] frameworks were considered. Jersey and Apache CXF are implementations of the JAX-RS specification [Hadley and Sandoz, 2008] that defines a set of Java APIs for the development of RESTful web services. Jersey is the reference implementation of JAX-RS, while Apache CXF is an open source services framework with a broader scope that has recently added support for JAX-RS. Though there are neither conceptual nor technological differences of some relevance, Restlet comes with a number of handy extensions for the development of the target application. Moreover, Restlet provides an extension for JAX-RS in case of required compatibility.

Reflecting on the development process, a REST framework such as Restlet helps to speed the translation into code of an abstract resource design. Indeed, the arrangement of routes and resources through URI templates in the implementation is almost straightforward. While URI templates can be very helpful to better organize resources and to simplify the handling of requests, they may introduce undesired coupling if employed in the client side [Webber et al., 2010, pp. 98–99], so client applications should treat URIs as opaque and rely instead on hypermedia to discover new links that make advance through the application state. It should be noted that resources act as mere *façade* elements, while the actual processing is performed by the business layer. In this regard, there are no constraints on using plain classes, a database backbone or other alternatives. What matters is to comply with the HTTP uniform interface semantics, e.g. sending back a representation of the target resource in case of a `GET` request. As a result, the tool registry class and the OWL reasoner were reused from former version of Ontoolsearch.

In comparison with RPC-based middlewares, supporting representation formats such as Atom involves more effort in order to encapsulate the information in the appropriate fields. In contrast, serialized objects are transferred between clients and servers in Java RMI without requiring any translation. However, this has a profound impact on interoperation due to tight data coupling. Note

[13] `http://rubyonrails.org/`
[14] `http://jersey.java.net/`
[15] `http://cxf.apache.org/`

that the combination of a standardized representation format plus the HTTP uniform interface allows the use of not purpose-built third-party clients. For instance, Firefox can be used to browse tool categories in Ontoolsearch (see Figure 3).

4 Comparison of Java RMI vs RESTful Ontoolsearch versions

In this section, former Java RMI version of Ontoolsearch is compared with the RESTful one depicted in section 3; in doing so, a feature analysis is performed to assess whether identified limitations in RMI Ontoolsearch are overcome in the RESTful version. The comparison is completed with a performance and scalability study through a benchmarking experiment.

4.1 Feature analysis

Feature analysis is an evaluation method of software tools and software development procedures [Kitchenham, 1996] that is typically employed to decide which product to choose in a structured and systematic way [Kitchenham, 1997]. Feature analysis is normally performed by an individual providing the assessment to a list of properties in candidate products. The main advantage of this evaluation method is its great flexibility since it does not impose many prerequisites upon the tools or methods to be evaluated and it can be performed to any required level of detail. Feature analysis is not without limitations, though; produced assessments inevitably carry some degree of subjectivity and there is a risk of inconsistency if different assessors have different interpretations.

In this case, a feature analysis evaluation was chosen with the aim of assessing whether identified limitations of the former RMI version of Ontoolsearch were tackled in the RESTful version. Note that performance and scalability are not considered here, but in subsection 4.2 with a benchmarking experiment. Thus, Table 1 shows the features enabled by the RESTful version of Ontoolsearch.

Features F1, F2 and F3 correspond to the new functionalities incorporated to the RESTful version (see section 3). Remarkably, the browsing functionality was easily achieved by exposing collections as resources. With respect to keyword queries, the well-known practice of encapsulating the query as part of the URI has been followed, while the actual processing is made by an existing text search engine.

Feature F4 is specially important to spread the user base of Ontoolsearch. In this regard, choosing a popular format such as Atom and adhering to HTTP's uniform interface enables the use of an extensive list of clients for accessing the functionalities of Ontoolsearch. In contrast, former version requires the usage of a special-purpose client as a result of the custom Java RMI interface.

Id	Feature	Comment
F1	Semantic search browsing	New functionalities
F2	Keyword-based queries	
F3	Tool browsing	
F4	Facilitate the use of third-party clients	Due to uniform interface and Atom
F5	Bookmarking of tools, searches...	Due to uniform interface and stateless communication
F6	Cacheable responses	
F7	Mashup capability	

Table 1: New features enabled by the RESTful version of Ontoolsearch.

With respect to features F5 and F6, resources such as tools and searches can be bookmarked and shared, e.g. by e-mail. Moreover, a resource representation such as a query result can be cached, thus improving efficiency by potentially avoiding some network interactions. These capabilities are due to the stateless communication constraint, i.e. the result of a call does not depend on previous requests, and to the uniform interface constraint since resources are univocally identified by URIs and expose an HTTP interface with standard semantics. In comparison, RPC-style infrastructures such as Java RMI only allow access to data elements through a remote interface. This fact along with the use of a stateful protocol severely limits the visibility of interactions, thus precluding the use of caching or bookmarking actions.

Finally, feature F7 refers to the creation of so-called web mashups [Benslimane et al., 2008]. As in the precedent case, the uniform interface and the stateless communication constraints enable, for instance, the integration ("mashing up") of the results provided by RESTful Ontoolsearch and other web searchers. Note that features F4, F6 and F7 can be combined in novel ways as illustrated in the following real example: an Atom reader such as Google Reader can be employed to add a subscription to the `SemQueryList` feed[16], thus serving to monitor semantic searches submitted to RESTful Ontoolsearch without writing a single line of code.

4.2 Benchmarking experiment

Beyond the feature analysis of subsection 4.1, the comparison of the former RMI version of Ontoolsearch versus the RESTful version is completed with a benchmarking experiment [Kitchenham, 1996]. The goal of this study is to compare the performance and scalability of these systems with respect to semantic search,

[16] `http://www.gsic.uva.es/ontoolsearch/semsearch/queries` (see Figure 2)

the basic functionality of Ontoolsearch and, at the same time, the most computationally demanding. In doing so, the methodology defined in [Jain, 1991, pp. 22-28] has been followed.

The experimental setup consists of a server and a client computers connected via a LAN. The client is a regular PC laptop while the server is a Linux virtual machine with 2 CPUs and 2 GB RAM virtualized with Eucalyptus[17], a software infrastructure for implementing private cloud computing. In this regard, it is possible to provision a dedicated machine with the desired configuration in a flexible and cost-effective way. Both Java RMI and RESTful versions of Ontoolsearch were deployed in the server machine along with the tool dataset composed of 103 instances. It should be noted that RMI Ontoolsearch requires HTTP tunneling to cope with firewalls, although it has a profound impact on performance ([Juric et al., 2004] reports an order of magnitude worse). Thus, in this study HTTP tunneling was disabled in order to facilitate the comparison of RPC-based middlewares versus RESTful approaches[18].

With respect to the metrics employed in the study, the response time and the rate of queries served were chosen. Response time is defined as the interval between a request and the system response; it is a common metric of raw performance that reflects the user's point of view. The second metric is defined as the rate of queries that can be serviced by the system; it is a throughput metric of special interest to administrators in order to assess the capacity and scalability of a system. The workload consisted of a simple program generating semantic queries of similar complexity, e.g. "any tool that supports communication tasks". The only workload parameter is the number of concurrent requests, implemented with threads. Thus, this program constitutes the client-side component of the experimental setup and is in charge of logging the measured results.

Measuring the response time of sequential requests, i.e. with no concurrency, RMI Ontoolsearch achieves 0.5 seconds versus 0.9 with RESTful Ontoolsearch. Such difference is due to the use of a binary protocol in the RMI case instead of a text protocol like HTTP plus the Atom encapsulation overhead. Moreover, network payload is smaller in RMI, since the binary RMI representation of a semantic query is $\sim$3 times smaller than the Atom representation.

When dealing with concurrency, Figure 5 shows that RESTful Ontoolsearch attains a better throughput than the RMI version; RESTful Ontoolsearch achieves a nominal capacity of 28.1 queries served per second versus 7.8 with RMI. Indeed, the RMI version collapses when more than 100 concurrent requests are submitted to the system. Analysing the mean response time, RESTful Ontoolsearch evolves gracefully: 1.4 seconds for 25 requests, 2.2 for 50, 3.4 for 100 and 5.5 for 200 (see Figure 6). In contrast, the RMI version exhibits correspondingly

[17] `http://www.eucalyptus.com/`

[18] For this reason the experiments were run in a LAN with no firewalls involved.

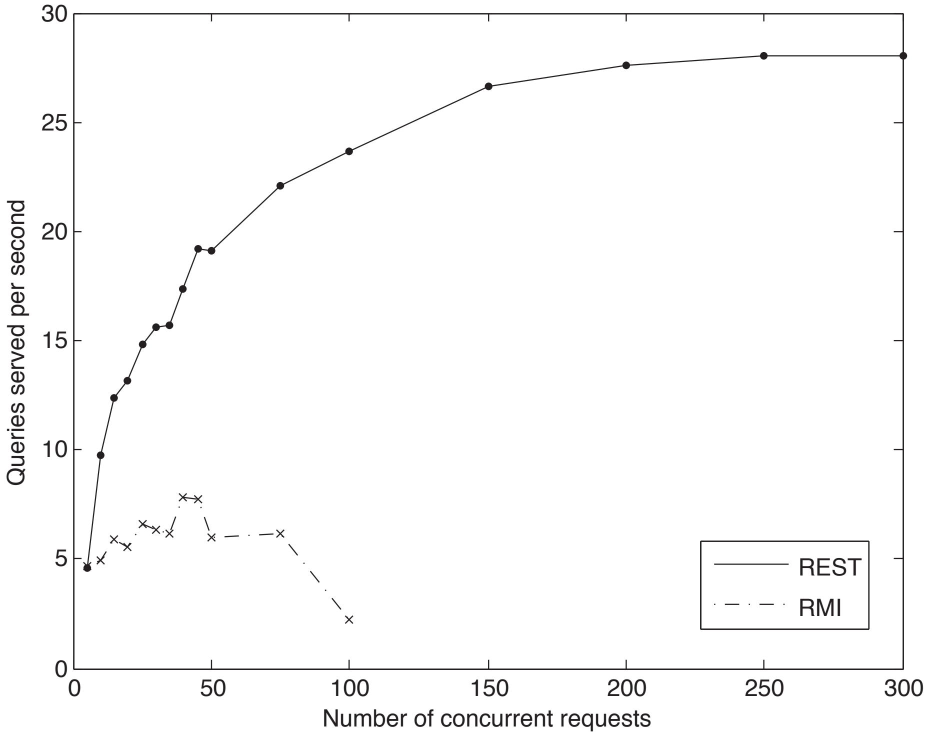

Figure 5: Rate of semantic queries served per second.

higher mean response times: 3.4 seconds for 25 requests, 8.0 for 50 and 41.3 for 100.

These results evidence that the scalability of RESTful Ontoolsearch is considerable better than in the RMI case. This is attributable to the stateless communication constraint of REST that allows the server component to quickly free resources. Although response time in sequential requests is better in RMI, it should be noted that the situation would be reversed if HTTP tunneling was enabled, as discussed above. Moreover, this study does not show the effect of caching because semantic queries involves the creation of resources and are thus not cacheable. Note, however, that all requests associated to the new functionalities of RESTful Ontoolsearch can be cached; given that caching is specially difficult to achieve with RPC-based middlewares [Vinoski, 2008a], significant REST performance improvements can be expected if cacheable requests were considered.

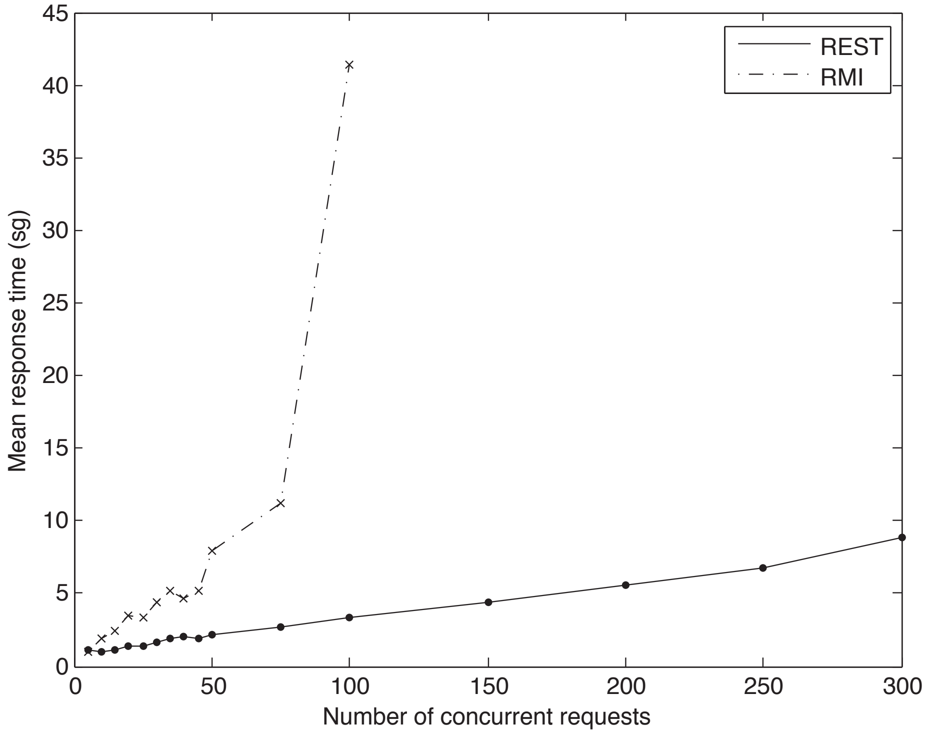

Figure 6: Mean response time as a function of the load.

5 Summary of lessons learned

The precedent sections presented the case study of developing a RESTful version of the Ontoolsearch system. Throughout this paper the implications of REST design and implementation have been thoroughly discussed. Furthermore, this case study have evidenced a number of REST benefits in comparison with RPC-style infrastructures. As a wrap-up of the previous discussions, Table 2 distills a set of lessons learned that may be useful for distributed systems designers.

The REST design block stresses the need of adopting a resource-centric viewpoint (lesson L1), with special care to the representation formats to be supported (lesson L2) and the connectedness of resources (lesson L3). In this regard, an understanding of REST constraints is recommended to avoid design pitfalls such as breaking the uniform interface by exposing an operation as a resource. Nevertheless, the authors found no big difficulties when developing RESTful Ontoolsearch in spite of their background on RPC-style systems.

Within the REST implementation block, lesson L4 recommends the use of a

Block	Id	Lesson learned
RESTful design	L1	Resources have to be designed with care, do not expose operations
	L2	Consider existing representation formats, taking into account your clients' needs
	L3	Connect resources with hyperlinks and check the course of events
RESTful implementation	L4	A RESTful framework may speed implementation
	L5	When RESTifying a legacy application the business layer may be reused
	L6	Supporting one (or more) representation format(s) requires some development effort
RPC vs REST architectural styles	L7	RPC performance may be better
	L8	Scalability gains expected in RESTful architectures
	L9	Increased interoperability in RESTful architectures
	L10	RESTful architectures enable caching, resource bookmarking and web mashups

Table 2: Summary of lessons learned.

REST framework; programming language, standards support, extensibility and documentation are some important criteria to be considered here. In the case of RESTful Ontoolsearch, the Restlet framework definitely contributed to speed up the development of the prototype by facilitating the translation of design to code and providing a number of useful extensions, e.g. Atom support. Furthermore, part of the business layer was reused from the previous RMI version, leading to lesson L5 which applies to the RESTification of legacy applications. To end this block, lesson L6 reminds that some development effort is required to comply with a representation format.

With respect to the comparison of REST and RPC architectural styles, lesson L7 warns that performance may be better with an RPC middleware as evidenced in the previous benchmarking study (see subsection 4.2). In this regard, a binary protocol usually performs better than a text protocol and the stateless communication constraint of REST may decrease network performance since repetitive data have to be sent again in a series of requests. Despite this, the use of caches has the potential to lessen the aforementioned effects in performance by eliminating some interactions. Considering scalability, a significant improvement is expected in RESTful architectures (lesson L8) since the server component does not have to store state between requests and caching also contributes to scalability. Indeed, this lesson is supported by the significant improvement of the scalability of RESTful Ontoolsearch in the benchmarking study. Finally, lessons L9 and L10 point out some benefits of RESTful architectures that are specially difficult to achieve with RPC approaches. As regards interoperability, standard data formats and the uniform interface constraint enable the use of application components provided by external independent organizations. Moreover, REST promotes the visibility of interactions by means of stateless communication and the uniform interface constraints, thus enabling caching, resource bookmarking and web mashups.

6 Conclusions and Future Work

The evolution of the Web into a computing platform has led to the emergence of a new breed of web applications ranging many different domains. What distinguishes them is the REST architectural style that enables the development of highly flexible, decoupled and scalable distributed applications. The disruptive nature of REST contrasts with the traditional RPC approach of the distribution systems community, leading to heated debates between REST and RPC proponents. However, there is a lack of comparisons of both approaches in the literature, as well as case studies exemplifying the development of a RESTful application.

This paper presented a revamped RESTful version of a legacy RPC-based search system of educational tools named Ontoolsearch. When designing the new system, differences between RPC and REST became evident; in this regard, resources are directly exposed to clients using the generic HTTP uniform interface that has standard semantics. In contrast, RPC enforces the exposure of custom interfaces, thus leading to restricted interoperability with third-party clients. Furthermore, limited visibility of interactions in RPC precludes monitoring or bookmarking submitted searches to the former version, hence the feature analysis that was carried out served to assess that all these limitations were overcome in the RESTful version. The comparison was completed with a benchmarking study, showing that response time is better in the RPC case, attributed to the use of a binary protocol instead of a text protocol and Atom encapsulation. Despite this, the scalability of RESTful Ontoolsearch is superior, measuring a peak throughput of 28.1 queries served per second versus 7.8 with the former version. A further outcome of this work is the set of lessons learned derived from this work that may be of special interest for redesigning legacy RPC-based systems suffering from similar limitations.

Future work includes the development of new functionalities for the annotation of tools with the aim of extending the tool dataset and spreading the creation of a community of teachers interested in the use of educational tools in the classroom. Furthermore, the integration of Ontoolsearch within the GLUE! architecture [Alario-Hoyos and Wilson, 2010] is planned to support the discovery and integration of external tools in Virtual Learning Environments. In addition, new case studies with different requirements should be carried out to further assess the benefits of the REST architectural style.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Innovation projects TIN2008-03023, TIN2011-28308-C03-02 and IPT-430000-2010-054, and the Autonomous Government of Castilla and León, Spain, projects VA293A11-2 and VA301B11-2.

References

[Adamczyk et al., 2011] Adamczyk, P., Smith, P., Johnson, R., and Hafiz, M. (2011). REST and Web Services: In Theory and in Practice. In [Wilde and Pautasso, 2011], pages 35–57.

[Alario-Hoyos and Wilson, 2010] Alario-Hoyos, C. and Wilson, S. (2010). Comparison of the main alternatives to the integration of external tools in different platforms. In *Proceedings of the third International Conference of Education, Research and Innovation(ICERI 2010)*, pages 3466–3476, Madrid, Spain.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge, UK.

[Benslimane et al., 2008] Benslimane, D., Dustdar, S., and Sheth, A. (2008). Services mashups: The new generation of web applications. *IEEE Internet Computing*, 12(5):13 –15.

[Berners-Lee et al., 2005] Berners-Lee, T., Fielding, R., and Masinter, L. (2005). Uniform resource identifier (URI): Generic syntax. Standard RFC 3986, The Internet Engineering Task Force (IETF).

[Bernstein, 1996] Bernstein, P. (1996). Middleware: a model for distributed system services. *Communications of the ACM*, 39(2):86–98.

[Coulouris et al., 2005] Coulouris, G., Dollimore, J., and Kindberg, T. (2005). *Distributed systems: concepts and design*. Addison-Wesley, Harlow, UK, fourth edition.

[Curbera et al., 2002] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S. (2002). Unraveling the Web Services Web. *IEEE Internet Computing*, 6(2):86–93.

[Fielding, 2000] Fielding, R. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine.

[Fielding et al., 1999] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol - HTTP/1.1. Draft Standard RFC 2616, The Internet Engineering Task Force (IETF).

[Fielding and Taylor, 2002] Fielding, R. T. and Taylor, R. N. (2002). Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150.

[Guha et al., 2003] Guha, R., McCook, R., and Miller, E. (2003). Semantic search. In *Proceedings of the Twelfth International World Wide Web Conference (WWW2003)*, pages 700–709, Budapest, Hungary. ACM.

[Hadley and Sandoz, 2008] Hadley, M. and Sandoz, P. (2008). Java API for RESTful Web Services. Specification JSR-000311, Sun Microsystems, Inc.

[Jain, 1991] Jain, R. (1991). *The art of computer systems performance analysis. Techniques for experimental design, measurement, simulation and modeling.* John Wiley, New York, NJ, USA.

[Juric et al., 2004] Juric, M. B., Kezmah, B., Hericko, M., Rozman, I., and Vezocnik, I. (2004). Java RMI, RMI tunneling and Web services comparison and performance analysis. *ACM SIGPLAN Notices*, 39:58–65.

[Kitchenham, 1996] Kitchenham, B. A. (1996). Evaluating software engineering methods and tools. Part 1: the evaluation context and evaluation methods. *SIGSOFT Software Eng. Notes*, 21(1):11–15.

[Kitchenham, 1997] Kitchenham, B. A. (1997). Evaluating software engineering methods and tools. Part 5: The influence of human factors. *SIGSOFT Software Eng. Notes*, 22(1):13–15.

[Larkin and Simon, 1987] Larkin, J. and Simon, H. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11(1):65–100.

[Maleshkova et al., 2010] Maleshkova, M., Pedrinaci, C., and Domingue, J. (2010). Investigating Web APIs on the World Wide Web. In *Proceedings of the 8th IEEE European Conference on Web Services (ECOWS2010)*, pages 107–114, Ayia Napa, Cyprus.

[Muller and Kuhn, 1993] Muller, M. J. and Kuhn, S. (1993). Participatory design. *Communications of the ACM*, 36(6):24–28.

[Nottingham and Sayre, 2005] Nottingham, M. and Sayre, R. (2005). The Atom Syndication Format. Standards Track RFC 4287, The Internet Engineering Task Force (IETF).

[OMG, 1991] OMG (1991). Common Object Request Broker: Architecture and Specification. Technical Report 1991/91-08-01, The Object Management Group.

[O'Reilly, 2005] O'Reilly, T. (2005). What is Web 2.0. Design patterns and business models for the next generation of software. URL: http://oreilly.com/web2/archive/what-is-web-20.html, last visited March 2011.

[Parastatidis et al., 2010] Parastatidis, S., Webber, J., Silveira, G., and Robinson, I. (2010). The Role of Hypermedia in Distributed System Development. In Pautasso, C., Wilde, E., and Marinos, A., editors, *Proceedings of the First International Workshop on RESTful Design (WS-REST 2010), co-located with the 19th International World Wide Web Conference (WWW2010)*, pages 16–22. Raleigh, NC, USA.

[Pautasso and Wilde, 2011] Pautasso, C. and Wilde, E. (2011). Introduction. In [Wilde and Pautasso, 2011], pages 1–18.

[Pautasso et al., 2008] Pautasso, C., Zimmermann, O., and Leymann, F. (2008). Restful web services vs. big web services: making the right architectural decision. In *Proceedings of the 17th International World Wide Web Conference (WWW2008)*, pages 805–814, Beijing, China. ACM.

[Raman, 2009] Raman, T. (2009). Toward 2W, beyond web 2.0. *Communications of the ACM*, 52(2):52–59.

[Richardson and Ruby, 2007] Richardson, L. and Ruby, S. (2007). *RESTful web services.* O'Reilly Media, Inc., Sebastopol, CA, USA.

[Richardson, 2010] Richardson, W. (2010). *Blogs, wikis, podcasts, and other powerful web tools for classrooms.* Corwin Press, Thousand Oaks, CA, USA, third edition.

[Sun, 1999] Sun (1999). Enterprise JavaBeans Specification. Technical report, Sun Microsystems Laboratories, Inc.

[Sun, 2004] Sun (2004). Java Remote Method Invocation Specification. Technical report, Sun Microsystems Laboratories, Inc.

[Vega-Gorgojo et al., 2010] Vega-Gorgojo, G., Bote-Lorenzo, M. L., Asensio-Pérez, J. I., Gómez-Sánchez, E., Dimitriadis, Y. A., and Jorrín-Abellán, I. M. (2010). Semantic search of tools for collaborative learning with the Ontoolsearch system. *Computers & Education*, 54(4):835–848.

[Vega-Gorgojo et al., 2008] Vega-Gorgojo, G., Bote-Lorenzo, M. L., Gómez-Sánchez, E., Asensio-Pérez, J. I., Dimitriadis, Y. A., and Jorrín-Abellán, I. M. (2008). Ontoolcole: Supporting educators in the semantic search of CSCL tools. *Journal of Universal Computer Science (JUCS)*, 14(1):27–58.

[Vinoski, 2008a] Vinoski, S. (2008a). Convenience over correctness. *IEEE Internet Computing*, 12(4):89–92.

[Vinoski, 2008b] Vinoski, S. (2008b). Serendipitous reuse. *IEEE Internet Computing*, 12(1):84–87.

[Waldo et al., 1994] Waldo, J., Wyant, G., Wollrath, A., and Kendall, S. (1994). A note on distributed computing. Technical Report SMLI TR-94-29, Sun Microsystems Laboratories, Inc.

[Webber et al., 2010] Webber, J., Parastatidis, S., and Robinson, I. (2010). *REST in Practice: Hypermedia and Systems Architecture*. O'Reilly Media, Inc., Sebastopol, CA, USA.

[Wilde and Pautasso, 2011] Wilde, E. and Pautasso, C., editors (2011). *REST: From Research to Practice*. Springer, New York, NY, USA.