

xOperator – Interconnecting the Semantic Web and Instant Messaging Networks

Sebastian Dietzold¹, Jörg Unbehauen², and Sören Auer^{1,3}

¹ Universität Leipzig, Department of Computer Science
Johannissgasse 26, D-04103 Leipzig, Germany,
dietzold@informatik.uni-leipzig.de

² Leuphana - University of Lüneburg, Faculty III Environmental Sciences and
Engineering, Volgershall 1, D-21339 Lüneburg
joerg@unbehauen.net

³ University of Pennsylvania, Department of Computer and Information Science
Philadelphia, PA 19104, USA,
auer@seas.upenn.edu

Abstract. Instant Messaging (IM) is in addition to Web and Email the most popular service on the Internet. With xOperator we present a strategy and implementation which deeply integrates Instant Messaging networks with the Semantic Web. The xOperator concept is based on the idea of creating an overlay network of collaborative information agents on top of social IM networks. It can be queried using a controlled and easily extensible language based on AIML templates. Such a deep integration of semantic technologies and Instant Messaging bears a number of advantages and benefits for users when compared to the separated use of Semantic Web technologies and IM, the most important ones being context awareness as well as provenance and trust. We showcase how the xOperator approach naturally facilitates contacts and calendar management as well as access to large scale heterogeneous information sources.

1 Introduction

With estimated more than 500 million users⁴ Instant Messaging (IM) is in addition to Web and Email the most popular service on the Internet. IM is used to maintain a list of close contacts (such as friends or co-workers), to synchronously communicate with those, exchange files or meet in groups for discussions. Examples of IM networks are ICQ, Skype, AIM or the Jabber protocol and network⁵. The latter is an open standard and the basis for many other IM networks such as Google Talk, Meebo and Gizmo.

While there were some proposals and first attempts to bring semantic technologies together with IM (e.g. [9, 5, 12]) in this paper we present a strategy and implementation called xOperator, which deeply integrates both realms in order

⁴ According to a sum up available at:

http://en.wikipedia.org/wiki/Instant_messaging#User_base

⁵ <http://www.jabber.org/>

to maximise benefits for prospective users. The xOperator concept is based on the idea of additionally equipping an users' IM identity with a number of information sources this user owns or trusts (e.g. his FOAF profile, iCal calendar etc.). Thus the social IM network is overlaid with a network of trusted knowledge sources. An IM user can query his local knowledge sources using a controlled (but easily extensible) language based on Artificial Intelligence Markup Language (AIML) templates[13]. In order to pass the generated machine interpretable queries to other xOperator agents of friends in the social IM network xOperator makes use of the standard message exchange mechanisms provided by the IM network. After evaluation of the query by the neighbouring xOperator agents results are transferred back, filtered, aggregated and presented to the querying user.

Such a deep integration of semantic technologies and IM bears a number of advantages and benefits for users when compared to the separated use of Semantic Web technologies and IM. From our point of view the two most crucial ones are:

- **Context awareness.** Users are not required to world wide uniquely identify entities, when it is clear what/who is meant from the context of their social network neighbourhood. When asked for the current whereabouts of Sebastian, for example, xOperator can easily identify which person in my social network has the name Sebastian and can answer my query without the need for further clarification.
- **Provenance and trust.** IM networks represent carefully balanced networks of trust. People only admit friends and colleagues to their contact list, who they trust seeing their online presence, not being bothered by SPAM and sharing contact details with. Overlaying such a social network with a network for semantic knowledge sharing and querying naturally solves many issues of provenance and trust.

The paper is structured as follows: after presenting envisioned usage scenarios and requirements in Section 2 we exhibit the technical xOperator architecture in Section 3. We report about a first xOperator evaluation according to different use cases in Section 4, present related work in Section 5. We draw conclusions and suggest directions for future work in Section 6.

2 Agent Communication Scenarios and Requirements

This section describes the three envisioned agent communication scenarios for xOperator. We will introduce some real-world application scenarios also later in Section 4. Figure 1 shows a schematic depiction of the communication scenarios. The figure is divided vertically into four layers.

The first two layers represent the World Wide Web. Mutually interlinked RDF documents (such as FOAF documents) reference each other using relations such as `rdf:seeAlso`.⁶ These RDF documents could have been generated

⁶ The prefix `rdf`, `rdfs`, `foaf` and `ical` used in this paper represent the well known namespaces (e.g. `http://xmlns.com/foaf/0.1/` for `foaf`).

manually, exported from databases or could be generated from other information sources. These can be, for example, mailing list archives which are represented as SIOC ontologies or personal calendars provided by public calendaring servers such as Google calendar. In order to make such information available to RDF aware tools (such as xOperator) a variety of transformation and mapping techniques can be applied. For the conversion of iCal calendar information for example we used Masahide Kanzaki's ical2rdf service⁷.

The lower two layers in Figure 1 represent the Jabber Network. Here users are interacting synchronously with each other, as well as users with artificial agents (such as xOperator) and agents with each. A user can pose queries in natural language to an agent and the agent transforms the query into one or multiple SPARQL queries. Thus generated SPARQL queries can be forwarded either to a SPARQL endpoint or neighbouring agents via the IM networks transport protocol (XMPP in the case of Jabber). SPARQL endpoints evaluate the query using a local knowledge base, dynamically load RDF documents from the Web or convert Web accessible information sources into RDF. The results of SPARQL endpoints or other agents are collected, aggregated, filtered and presented to the user depending on the query as list, table or natural language response.

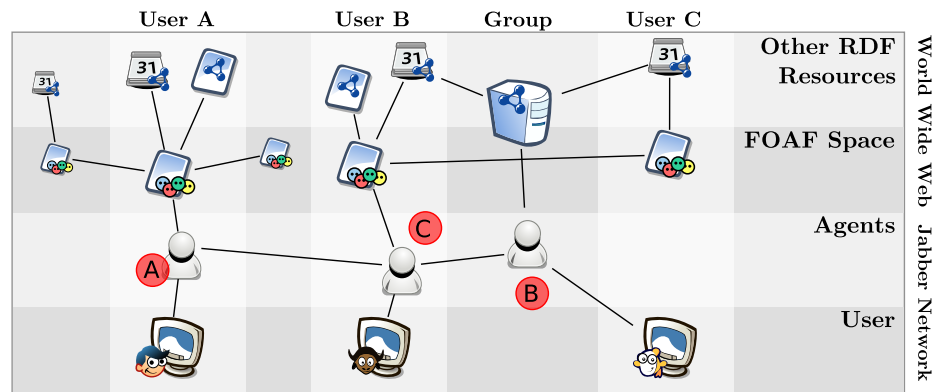


Fig. 1. Agent communication scenarios: (a) personal agent, (b) group agent, (c) agent network.

The different communication scenarios are described in the following subsections:

2.1 Personal Agent (A)

This scenario is the most important one and also builds the foundation for the other two communication scenarios. A user of an Instant Messaging network

⁷ <http://www.kanzaki.com/courier/ical2rdf>

installs his own personal agent and configures information sources he owns or trusts. For easy deployment the software representing the agent could be distributed together with (or as a plugin of) the IM network client (e.g. Pidgin). Information sources can be for example a FOAF profile of the user containing personal information about herself and about relationships to other people she knows and where to find further information about these. This information is represented in FOAF using the properties `foaf:knows` and `rdfs:seeAlso`. The following listing shows an excerpt from a FOAF profile.

```
:me a foaf:Person ;
  foaf:knows [
    a foaf:Person ;
    rdfs:seeAlso <http://eye48.com/foaf.rdf> ;
    foaf:name "Michael Haschke" ;
    foaf:nick "Haschek" ] .
```

Additionally this FOAF profile can link to other RDF documents which contain more information about the user and his activities. The RDF version of his calendar, for example, could be linked as follows:

```
:me rdfs:seeAlso <http://.../ical2rdf?u=http...> .
<http://.../ical2rdf?u=http...> a ical:Vcalendar ;
  rdfs:label "Haschek's Calendar" .
```

Such links span a network of information sources as depicted in Figure 1. Each user maintains his own information and links to information sources of his acquaintances. Depending on the query, the agent will access the respective resources. The following example queries are possible, when FOAF profiles are known to the agent: Tell me the phone / homepage / ... of Frank! What is the birthday of Michael? Where is Dave now? Who knows Alex?

2.2 Group Agent (B)

This communication scenario differs from the Personal Agent scenario in that multiple users get access to the same agent. The agent should be able to communicate with multiple persons at the same time and to answer queries in parallel. As is also depicted in Figure 1 the agent furthermore does not only access remote documents but can also use a triple store for answering queries. When used within a corporate setting this triple store can for example contain a directory with information about employees or customers. The triple store can be also used to cache information obtained from other sources and thus facilitates faster query answering. For agents themselves, however, the distinction between RDF sources on the Web and information contained in a local triple store is not relevant.

2.3 Agent Network (C)

This scenario extends the two previous ones by allowing communication and interaction between agents. The rationale is to exploit the trust and provenance

characteristics of the Instant Messaging network: Questions about or related to acquaintances in my network of trust can best be answered by their respective agents. Hence, agents should be able to talk to other agents on the IM network. First of all, it is crucial that agents on the IM network recognize each other. A personal agent can use the IM account of its respective owner and can access the contact list (also called roster) and thus a part of its owner’s social network. The agent should be able to recognise other personal agents of acquaintances in this contact list (auto discovery) and it should be possible for agents to communicate without interfering with the communication of their owners. After other agents are identified it should be possible to forward SPARQL queries (originating from a user question) to these agents, collect their answers and present them to the user.

3 Technical Architecture

First of all, the xOperator agent is a mediator between the Jabber Instant Messaging network⁸ on one side and the World Wide Web on the other side. xOperator is client in both networks. He communicates anonymously (or using configured authentication credentials) on the WWW by talking HTTP with Web servers. On the Jabber network xOperator utilizes the Extensible Messaging and Presence Protocol (XMPP, [11]) using the Jabber account information provided by its owner. Jabber clients only communicate with the XMPP server associated with the user account. Jabber user accounts are have the same syntax as email addresses (e.g. `soerenauer@jabber.ccc.de`). The respective Jabber server cares about routing messages to the server associated with the target account or temporarily stores the message in case the target account is not online or its server is not reachable. Since 2004 XMPP is a standard of the Internet Engineering Task Force and is widely used by various services (e.g. Google Talk). Figure 2 depicts the general technical architecture of xOperator.

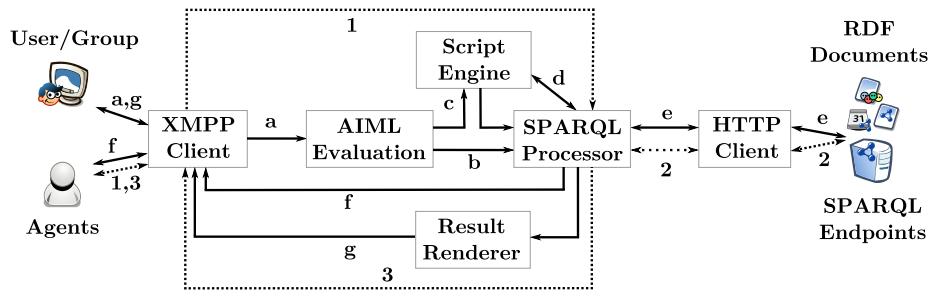


Fig. 2. Technical architecture of xOperator.

⁸ Our implementation currently supports the Jabber network, but can be easily extended to other IM networks such as Skype, ICQ or MSN Messenger

The agent works essentially in two operational modi:

1. (Uninterrupted line) Answer natural language questions posed by a user using SPARQL queries and respond to the user in natural language according to a predefined template. Questions posed by a user (a) are either directly mapped to a SPARQL query template (b) or SPARQL queries are generated by a query script (c), which might obtain additional information by means of sub queries (d). The resulting SPARQL query will be evaluated on resources of the user (e), as well as passed on the Jabber network to neighbouring agents for evaluation (f). All returned results are collected and prepared by a result renderer for presentation to the user (g). Algorithm 1 demonstrates the workings of xOperator.

Algorithm 1: Evaluation of XMPP user input.

Input: User input I from XMPP
Output: Sendable Agent response
Data: set $S = A \cup D \cup E$ of agents, documents and endpoints
Data: set C of AIML categories
Data: set $R = \emptyset$ of results

```
1 if  $I$  is an admin or extension command then return executeCommand ( $I$ )
2 else if  $I$  has no match in  $C$  then return defaultmsg
3 else if  $I$  has standard match in  $C$  then return aimlResult ( $I, C$ )
4 else
5   | if  $I$  has SPARQL template match in  $C$  then
6   |   | Query = fillPatterns (aimlResult ( $I, C$ ))
7   | else if  $I$  has query script match in  $C$  then
8   |   | Query = runScript (aimlResult ( $I, C$ ))
9 if Query then
10  | foreach  $s \in S$  do
11  |   |  $R = R \cup$  executeQuery(Query,  $s$ )
12  |   return renderResults ( $R$ );
13 else
14  | return error
```

2. (Dotted line) Receive SPARQL queries from neighbouring agents (1) on the IM network, evaluate these queries (2) on the basis of locally known RDF documents and SPARQL endpoints and send answers as XML SPARQL Result Set [3] back via XMPP (3).

In both cases the agent evaluates SPARQL queries by querying a remote SPARQL endpoint via HTTP GET Request according to the SPARQL HTTP Bindings [4] or by retrieving an RDF document as well via HTTP and evaluating the query by means of a local SPARQL query processor.

In the following we describe first the natural language component on the basis AIML templates and address thereafter the communication in the Jabber network.

3.1 Evaluation of AIML Templates

The Artificial Intelligence Markup Language (AIML, [13]) is an XML dialect for creating natural language software agents. In [6] the authors describe AIML to enable pattern-based, stimulus-response knowledge content to be served, received and processed on the Web and offline in the manner that is presently possible with HTML and XML. AIML was designed for ease of implementation, ease of use by newcomers, and for interoperability with XML and XML derivatives such as XHTML. Software reads the AIML objects and provides application-level functionality based on their structure. The AIML interpreter is part of a larger application generically known as a bot, which carries the larger functional set of interaction based on AIML. A software module called a responder handles the human-to-bot or bot-to-bot interface work between an AIML interpreter and its object(s). In xOperator AIML is used for handling the user input received through the XMPP network and to translate it into either a query or a call to a script for more sophisticated evaluations.

The most important unit of knowledge in AIML is the category. A category consists of at least two elements, a pattern and a template element. The pattern is evaluated against the user input. If there is a match, the template is used to produce the response of the agent. It is possible to use the star (*) as a placeholder for any word in a pattern. We have extended this basic structure in two ways:

Simple Query Templates: In order to enable users to create AIML categories on the fly we have created an extension of AIML. It allows to map natural language patterns to SPARQL query templates and to fill variables within those templates with parameters obtained from *-placeholders in the natural language patterns.

```
<category>
  <pattern>TELL ME THE PHONE OF *</pattern>
  <template>
    <external name="query"
      param="SELECT DISTINCT ?phone WHERE {...}" />
  </template>
</category>
```

Within the SPARQL template variables in the form of `%n%` refer to *-placeholder (`n` refers to the n^{th} *-placeholder in the category pattern). The question for the phone number of a person, for example, can be represented with the following AIML template:

```
TELL ME THE PHONE OF *
```

A possible (very simple) SPARQL template using the FOAF vocabulary could be stored within the AIML category as follows:

```
SELECT DISTINCT ?phone WHERE
  { ?s foaf:name "%1%". ?s foaf:phone ?phone. }
```

On activation of a natural language pattern by the AIML interpreter the corresponding SPARQL templates variables are bound to the values of the placeholders and the resulting query is send independently to all known SPARQL endpoints and neighbouring agents. These answer independently and deliver result sets, which can complement each other, contain the same or contradictory results. The agent renders results as they arrive to the user, but filters duplicates and marks contradictory information. The agent furthermore annotates results with regard to their provenance.

This adoption of AIML is easy to use and directly extensible via the Instant Messaging client (cf. Sec. 3.2). However, more complex queries, which for example join information from multiple sources are not possible. In order to enable such queries we developed another AIML extension, which allows the execution of query scripts.

Query Scripts: Query scripts basically are small pieces of software, which run in a special environment where they have access to all relevant subsystems. They are given access to the list of known data sources and neighbouring agents. xOperator, for example, allows the execution of query scripts in the Groovy scripting language for Java. The execution of a query script results in the generation of a SPARQL query, which is evaluated against local information sources and passed to other agents as described in the previous section. We motivate and illustrate the workings of query scripts using an application scenario based on the FOAF space (cf. Figure 1). The FOAF space has the following characteristics:

- The `foaf:knows` relation points to other people known by this person.
- Other FOAF and RDF documents are linked through `rdfs:seeAlso`, allowing bots and agents to crawl through the FOAF space and to gather additional RDF documents like calendars or blog feeds.

To enable the agent to retrieve and evaluate additional information from sources which are referenced from the user's FOAF profile, a query script can contain subqueries, whose results are used within another query. Query scripts also enable the usage of special placeholders such as `now` or `tomorrow`, which can be populated for the querying of iCal calendars with the concrete values.

In order to extend the agent for other application domains or usage scenarios, xOperator allows to dynamically assign new query scripts to AIML categories. A query script is assigned to an AIML template by means of an `external` tag (as are also simple SPARQL templates). An example script implementing a subquery to retrieve relevant resources about a `foaf:person` is presented in Section 4.

3.2 Administration and Extension Commands

Users can easily change the configuration of their agents by using a set of administration and extension commands. These commands have a fix syntax and are executed without the AIML engine:

- `list ds, add ds {name} {uri}, del ds {name}`: With these commands, users can manage their trusted data sources. Each source is locally identified by a name which is associated to an URL.
- `list templates, add template {pattern} {query}, del template {pattern}`: With these template commands, users can manage simple query templates which are associated by its AIML pattern.
- `query {sparql query}`: This command is used to send on-the-fly SPARQL queries to the xOperator. The query will be evaluated on every datastore and routed to every agent in the neighbourhood. The query results will be rendered by a default renderer.
- `list ns, add ns {prefix} {uri}, del ns {prefix}`: To easlily create on-the-fly SPARQL queries, users can manage namespaces. The namespaces will be added to the namespace section in the on-the-fly query.
- `help`: This is an entry point for the help system.

3.3 XMPP Communication and Behaviour

While the HTTP client of the agent uses standard HTTP for querying SPARQL endpoints and the retrieval of RDF documents, we extended XMPP for the mutual communication between the agents. This extension complies with standard extension routines of XMPP will be ignored by other agents. With regard to the IM network the following functionality is required:

- The owner of the agent should be able to communicate easily with the agent. He should be able to manage the agent using the contact list (roster) and the agent should be easily recognizable.
- The agent has to have access to the roster of its owner in order to identify neighbouring agents.
- It should be possible for other agents to obtain information about the ownership of an agent. His requests will not be handled by other agents for security reasons if he can not be clearly assigned to an owner.
- The agent should be only visible for his owner and neighbouring agents (i.e. agents of contacts of his owner) and only accept queries from these.

As a consequence from those requirements it is reasonable that the agent acts using the account of its owner (main account) for the communication with other agents, as well as an additional account (proxy account) for the communication with its owner⁹. Due to the usage of the main account other agents can trust the agents answers and easily track the provenance of query results. Figure 3

⁹ Technically, it is sufficient for the agent to use the owner's account which, however, could create confusing situations for the user when communicating with 'herself'.

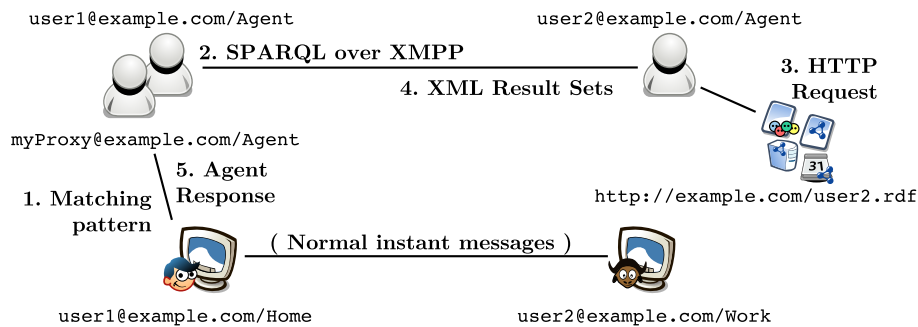


Fig. 3. XMPP Communication example

depicts the concept of using two different accounts for the communication with the owner and other agents. For unique identification of senders and recipients so called resource names (in the figure Home, Work and Agent) are used and simply appended to the account name.

We demonstrate the agent communication with two XMPP messages:

Agent Autodiscovery: Goal of the autodiscovery is the identification of agents among each other. For that purpose each agent sends a special message of type info/query (iq) to all known and currently active peers. Info/query messages are intended for internal communication and queries among IM clients without being displayed to the human users. An autodiscovery message between the two agents from Figure 3, for example, would look as follows:

```
<iq from="user1@example.com/Agent" type='get'
  to="user2@example.com/Agent" id='...'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

A positive response to this feature discovery message from an xOperator agent would contain a feature with resource ID <http://www.w3.org/2005/09/xmpp-sparql-binding>. This experimental identifier/namespace was created by Dan Brickley for SPARQL / XMPP experiments (cf. Section 5). The response message to the previous request would look as follows:

```
<iq from='user2@example.com/Agent' type='result'
  to='user1@example.com/Agent' id='...' />
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
      category='client' name='xOperator' type='bot'/>
    <feature
      var='http://www.w3.org/2005/09/xmpp-sparql-binding'/>
    <!-- ... more here -->
  </query>
</iq>
```

Similar XMPP messages are used for sending SPARQL queries and retrieving results. The latter are embedded into a respective XMPP message according to the SPARQL Query Results XML Format¹⁰.

Routing and Recall. Queries are propagated to all neighbouring xOperator agents. As currently there is no way of anticipating which agent could answer a question, asking all directly connected agents offers the best compromise between load and recall. Flooding the network beyond adjacent nodes would cause excessive load. Especially in the domain of personal information, persons or their respective agents directly related to the querying person or agent should be most likely to answer the query.

4 Evaluation

The xOperator concept was implemented in Java and is available as open-source software from: <http://aksw.org/Projects/xOperator>. The agent is able to log into existing accounts and can receive querying and configuration commands.

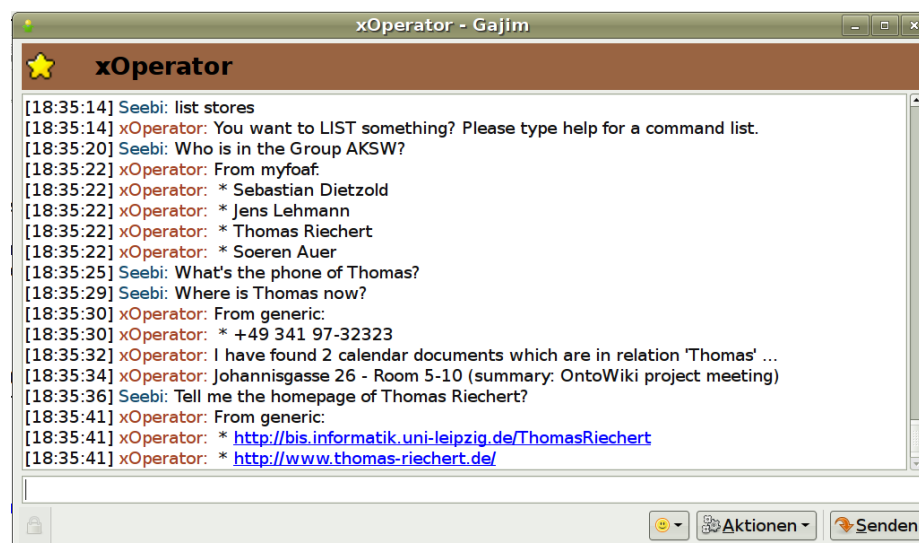


Fig. 4. Communication with the xOperator agent by means of an ordinary Jabber client.

We evaluated our approach in a number of scenarios, which included various heterogeneous information sources and a different number of agents. As information sources we used FOAF profiles (20 documents, describing 50 people), the

¹⁰ <http://www.w3.org/TR/rdf-sparql-XMLres/>

SPARQL endpoint of our semantic Wiki OntoWiki [2] (containing information about publications and projects), information stored in the LDAP directory service of our department, iCal calendars of group members from Google calendar (which are accessed using iCal2RDF) and publicly available SPARQL endpoints such as DBpedia [1]. Hence the resulting information space contains information about people, groups, organizations, relationships, events, locations and all information contained in the multidomain ontology DBpedia. We created a number of AIML categories, interacting with this information space. Some example patterns and corresponding timings for obtaining answers from the agent network in the three different network scenarios (personal agent, agent network and group agent) are summarized in Table 1.

	Template	Scenario 1	Scenario 2	Scenario 3
1	What is / Tell me (the) * of *	2.3	3.9	1.5
2	Who is member of *	3.5	4.3	1.6
3	Tell me more about *	3.2	5.6	1.1
4	Where is * now:	5.1	6.7	4.2
5	Free dates * between * and *	5.1	6.8	4.7
6	Which airports are near *	-	-	3.4

Table 1. Average response time in seconds (client to client) of some AIML patterns used in three scenarios: (1) 20 documents linked from one FOAF profile, 1 personal agent with no neighbourhood (2) 20 documents linked from different FOAF profiles and spread over a neighbourhood of 5 agents (3) one SPARQL endpoint as an interface to a Semantic Wiki or DBpedia store with one group agent

The first three templates represent queries which are answered using simple SPARQL templates. Template 4 makes use of a reserved word (`now`), which is replaced for querying with an actual value. Template 5 is implemented by means of a query script which retrieves all available time slots from the calendars of a group of people and calculates the intersection thus offering suitable times to arrange meetings or events, where the attendance of all group members is required. Template 6 uses the DBpedia SPARQL endpoint in a group agent setting to answer questions about the geographic location of places (such as airports). These query templates are meant to give some insights in the wealth of opportunities for employing xOperator. Further, AIML templates can be created easily, even directly from within the IM client (using the administration and extension commands as presented in Section 3.2).

A typical user session showing the communication with the agent is depicted in Figure 4. The response timings indicate that the major factor are latency times for retrieving RDF documents or querying SPARQL endpoints. The impact of the number of agents in the agent network as well as the overhead required by the xOperator algorithm is rather small. The timings are furthermore upper bounds, since answers are presented to the user as they arrive. This results in

intuitive perception that xOperator is a very responsive and efficient way for query answering.

Experiences during the evaluation have led to the following rules for creating patterns and queries in xOperator.

(1) *Query as fuzzy as possible:* Instant Messaging is a very quick means of communication. Users usually do not capitalize words and use many abbreviations. This should be considered, when designing suitable AIML patterns. If information about the person ‘Sören Auer’ should be retrieved, this can be achieved using the following graph pattern: `?subject foaf:name "Auer"`. However, information can be represented in multiple ways and often we have to deal with minor misrepresentations (such as trailing whitespace or wrong capitalizations), which would result for the above query to fail. Hence, less strict query clauses should be used instead. For the mentioned example the following relaxed SPARQL clause, which matches also substrings and is case insensitive, could be used:

```
?subject foaf:name ?name.  
FILTER regex(?name, '.*Auer.*', 'i')
```

(2) *Use patterns instead of qualified identifiers for properties:* Similar, as for the identification of objects, properties should be matched flexible. When searching for the homepage of ‘Sören Auer’ we can add an additional property matching clause to the SPARQL query instead of directly using, for example, the property identifier `foaf:homepage`):

```
?subject ?slabel ?spattern.  
?subject ?property ?value.  
?property ?plabel ?ppattern.  
FILTER regex(?spattern, '.*Auer.*', 'i')  
FILTER regex(?ppattern, '.*homepage.*', 'i')
```

This also enables multilingual querying if the vocabulary contains the respective multilingual descriptions. Creating fuzzy queries, of course, significantly increases the complexity of queries and will result in slower query answering by the respective SPARQL endpoint. However, since we deal with a distributed network of endpoints, where each one only stores relatively small documents this effect is often negligible.

(3) *Use sub queries for additional documents:* In order to avoid situations where multiple agents retrieve the same documents (which is very probable in a small worlds scenario with a high degree of interconnectedness) it is reasonable to create query scripts, which only distribute certain tasks to the agent network (such as the retrieval of prospective information sources or document locations), but perform the actual querying just once locally.

5 Related Work

Proposals and first prototypes which are closely related to xOperator and inspired its development are Dan Brickley's JQbus¹¹ and Chris Schmidt's SPARQL over XMPP¹². However, both works are limited to the pure transportation of SPARQL queries over XMPP.

Quite different but the xOperator approach nicely complementing are works regarding the semantic annotation of IM messages. In [9] for example the authors present a semantic archive for XMPP instant messaging which facilitates search in IM message archives. [5] suggests ways to make IM more semantics aware by facilitating the classification of IM messages, the exploitation of semantically represented context information and adding of semantic meta-data to messages. Comprehensive collaboration frameworks which include semantic annotations of messages and people, topics are, for example, CoAKTinG [12] and Haystack [7]. The latter is a general purpose information management tool for end users and includes an instant messaging component, which allows to semantically annotate messages according to a unified abstraction for messaging on the Semantic Web[10].

In [8] natural language interfaces (NLIs) are used for querying semantic data. The NLI used in xOperator employs only a few natural language processing techniques, like stop word removal for better template matching. Generic templates would be possible to define, but as [8] shows user interaction is necessary for clarifying ambiguities. For keeping IM conversation as simple as possible, domain specific templates using AIML were chosen. Finally, in [6] the author enhanced AIML bots by generating AIML categories from RDF models. Different to xOperator, these categories are static and represent only a fixed set of statements.

6 Conclusions and Future Work

With the xOperator concept and its implementation, we have showed how a deeply and synergistic coupling of Semantic Web technology and Instant Messaging networks can be achieved. The approach naturally combines the well-balanced trust and provenance characteristics of IM networks with semantic representations and query answering of the Semantic Web. The xOperator approach goes significantly beyond existing work which mainly focused either on the semantic annotation of IM messages or on using IM networks solely as transport layers for SPARQL queries. xOperator on the other hand overlays the IM network with a network of personal (and group) agents, which have access to knowledge bases and Web resources of their respective owners. The neighbourhood of a user in the network can be easily queried by asking questions in a subset of natural language. By that xOperator resembles knowledge sharing and exchange in offline communities, such as a group of co-workers or friends. We have showcased how the xOperator approach naturally facilitates contacts and

¹¹ <http://svn.foaf-project.org/foaftown/jqbus/intro.html>

¹² <http://crschmidt.net/semweb/sparqlxmpp/>

calendar management as well as access to large scale heterogeneous information sources. In addition to that, its extensible design allows a straightforward and effortless adoption to many other application scenarios such as, for example, sharing of experiment results in Biomedicine or sharing of account information in Customer Relationship Management.

In addition to adopting xOperator to new domain application we view the xOperator architecture as a solid basis for further technological integration of IM networks and the Semantic Web. This could include adding light-weight reasoning capabilities to xOperator or the automatic creation of AIML categories by applying NLP techniques. A more fine grained access control will be implemented in a future version. Instead of simply trusting all contacts on the roster, individual and group based policies can be created. An issue for further research is the implementation of a more sophisticated routing protocol, that allows query traversal beyond directly connected nodes without flooding the whole network.

References

1. Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. DBpedia: A Nucleus for a Web of Open Data. In *Proc. of ISWC/ASWC*, pages 722–735, 2007.
2. Sören Auer, Sebastian Dietzold, and Thomas Riechert. OntoWiki - A Tool for Social, Semantic Collaboration. In *Proc. of ISWC*, pages 736–749, 2006.
3. Dave Beckett and Jeen Broekstra. SPARQL Query Results XML Format. W3C Candidate Recommendation, World Wide Web Consortium (W3C), April 2006.
4. Kendall Grant Clark. SPARQL Protocol for RDF. W3C Recommendation, World Wide Web Consortium (W3C), 2007.
5. Thomas Franz and Steffen Staab. SAM: Semantics Aware Instant Messaging for the Networked Semantic Desktop. In *Semantic Desktop Workshop at the ISWC*, 2005.
6. Eric Freese. Enhancing AIML Bots using Semantic Web Technologies. In *Proc. of Extreme Markup Languages*, 2007.
7. David R. Karger, Karun Bakshi, David Huynh, Dennis Quan, and Vineet Sinha. Haystack: A General-Purpose Information Management Tool for End Users Based on Semistructured Data. In *Proc. of CIDR*, pages 13–26, 2005.
8. Esther Kaufmann and Abraham Bernstein. How useful are natural language interfaces to the semantic web for casual end-users? In *6th International Semantic Web Conference (ISWC 2007)*, pages 281–294, 2007.
9. Frank Osterfeld, Malte Kiesel, and Sven Schwarz. Nabu - A Semantic Archive for XMPP Instant Messaging. In *Semantic Desktop Workshop at the ISWC*, 2005.
10. Dennis Quan, Karun Bakshi, and David R. Karger. A Unified Abstraction for Messaging on the Semantic Web. In *WWW (Posters)*, 2003.
11. P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920, The Internet Engineering Task Force (IETF), October 2004.
12. Simon Buckingham Shum, David De Roure, Marc Eisenstadt, Nigel Shadbolt, and Austin Tate. CoAKTinG: Collaborative advanced knowledge technologies in the grid. In *Proc. of 2. Workshop on Adv. Collab. Env. at the HPDC-11*, 2002.
13. Richard Wallace. Artificial Intelligence Markup Language (AIML). Working draft, A.L.I.C.E. AI Foundation, 2005. 18 February 2005.