TripCom

Triple Space Communication

FP6 – 027324

Deliverable

D6.4

Component integration, demonstration, and documentation

Michael Lafite
Christian Schreiber
Francesco Corcoglioniti
Alessio Carenini
Philipp Obermeier
Sebastian Dill

March 31, 2009

Executive Summary

This document describes the prototype implementation realized as part of the TripCom project. The focus of the document is on the usage of the prototype. This means that the implemtation is described from the client perspective. A detailed description of implemenation and the architecture of the prototype can be found in [8].

The document gives a brief overview over the triplespace and the API which is exported by the implementation. Additionally, non-functional features, such as security and transaction, are described and a brief overview of the prototype architecture and the components is provided.

The second part of the deliverable provides a detailed description of the installation and configuration process of the prototype and a description of the use case implementations which have been created to demonstrate the advantages of triplespace computing.

Document Information

IST Project Number	FP6 – 027324	**Acronym**	TripCom
Full Title	Triple Space Communication		
Project URL	http://www.tripcom.org/		
Document URL			
EU Project Officer	Werner Janusch		

Deliverable	**Number**	6.4	**Title**	Component integration, demonstration, and documentation
Work Package	**Number**	6	**Title**	Triple Space Architecture and Component Integration

Date of Delivery	**Contractual**	M36	**Actual**	M36
Status	version 1.0		final ☒	
Nature	prototype ☒ report ☐ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Michael Lafite			
Resp. Author	Michael Lafite		**E-mail**	michael.lafite@complang.tuwien.ac.at
	Partner	TUW	**Phone**	(xxx) xxxx-xxx

Abstract (for dissemination)	This deliverable describes the prototype implementation realized as part of the TripCom project. The focus of the document is on the usage of the prototype. This means that the implemtation is described from the client perspective. Additionally, the deliverable provides a detailed description of the steps necessary to install and use the triple space.
Keywords	Triple Space, implementation, architecture, documentation

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
2009-03-09	1	Michael Lafite	Initial structure
2009-03-20	2	Sebastian Dill	Chapters 2
2009-03-20	3	Philipp Obermeier	Chapters 2
2009-03-30	4	Christian Schreiber	Chapters 3, 2
2009-03-30	5	Michael Lafite	Chapters 2
2009-03-31	6	Christian Schreiber	Introduction

Project Consortium Information

Acronym	Partner	Contact
Semantic Technology Institute Innsbruck http://www.sti-innsbruck.at	STI	Prof. Dr. Dieter Fensel Semantic Technology Institute (STI) Innsbruck, Austria E-mail: dieter.fensel@sti-innsbruck.at
National University of Ireland, Galway http://www.deri.ie	NUIG	Dr. Laurentiu Vasiliu Digital Enterprise Research Institute (DERI) Galway, Ireland Email: laurentiu.vasiliu@deri.org
University of Stuttgart http://www.iaas.uni-stuttgart.de/	USTUTT	Prof.Dr. Frank Leymann Inst. für Architektur von Anwendungssystemen (IAAS) Stuttgart, Germany E-mail: frank.leymann@informatik.uni-stuttgart.de
Vienna university of Technology http://www.complang.tuwien.ac.at/	TUW	Prof.Dr. eva Kühn Institut für Computersprachen Vienna, Austria E-mail: eva@complang.tuwien.ac.at
Free University Berlin http://www.ag-nbi.de/	FUB	Prof. Dr.-Ing. Robert Tolksdorf AG Netzbasierte Informationssysteme Berlin, Germany E-mail : tolk@inf.fu-berlin.de
Ontotext Lab, Sirma Group Corp. http://www.ontotext.com/	ONTO	Atanas Kiryakov, Vassil Momtchev, Ontotext Lab, Sirma Group Corp. Sofia, Bulgaria E-mail: vassil.momtchev@ontotext.com
Profium OY http://www.profium.com/	Profium	Dr. Janne Saarela Profium OY Espoo, Finland E-mail: janne.saarela@profium.com
CEFRIEL SCRL. http://www.cefriel.it/	CEFRIEL	Davide Cerri CEFRIEL SCRL. Milano, Italy E-mail: cerri@cefriel.it
Telefonica I+D http://www.tid.es/	TID	Noelia Pérez Crespo Telefonica I+D Madrid, España E-mail: npc@tid.es

Table of Contents

1 Introduction

This document describes the prototype implementation realized as part of the TripCom project. The focus of the document is on the usage of the prototype. This means that the implemtation is described from the client perspective. A detailed description of implemenation and the architecture of the prototype can be found in [8].

The document gives a brief overview over the triplespace and the API which is exported by the implementation. Additionally, non-functional features, such as security and transaction, are described and a brief overview of the prototype architecture and the components is provided.

The second part of the deliverable provides a detailed description of the installation and configuration process of the prototype and a description of the use case implementations which have been created to demonstrate the advantages of triplespace computing.

2 Using the Triple Space

This sections explains what the Triple Space can do from a client perspective, including the concepts of triples, spaces and sub-spaces and a brief description of space-based coordination principles (main operations RD, OUT, IN).

The TripCom prototype realizes the concept of a triple space. A triple space is a (virtual) shared memory which can be used by concurrent clients to store and retrieve triples. In TripCom these triples are represented by RDF. In the prototype the space can be structured into several subspaces which again can have subspaces. The API of the prototype (c.f. section 2.1) provides methods to put data into the space to get data from the space. Since a space can have subspaces, a method is provided which allows for retrieving data from the space and all its subspaces (read recursive). In addition there are methods to get data from the space without modifying the existing (read) and methods to consume (destructive read, in) data which removes the returned data. These methods and the other triple space methods described in section 2.1 can be used by clients to communicate with each other. Instead of sending triples directly between the clients, they communicate by means of exchanging triples over the space. Additionally, the clients can use the data in the space to coordinate their execution. Clients can use templates to specificy in which triples they are interested when retrieving operations are executed. In the current implementation these templates can be SPARQL queries. The only restriction is that the SPARQL query has to return a full triple (if this is not the case one has to use CONSTRUCT instead of SELECT).

2.1 API Description

In this section we will specify the interface capabilities a kernel provides to clients and other kernels for interaction (Section 2.1.1). Apart from that, we will define the API operations kernel provides for administrative tasks (Section 2.1.2).

2.1.1 Triple Space API

When considering the component interaction in a Triple Space kernel, it is important to identify the coordination patterns to be supported by the kernel as each pattern may need to be implemented by a different set of interactions between the internal components. These coordination patterns can be specified on the basis of the public *Triple Space API*.

The Triple Space API has been revised since the publication of the versions in [] and [8] as a result of further discussions, insights won through implementation and the need to provide different configurations of the API to account for scalability, security and expressivity trade-offs. The revised API is given in this section. It allows for three levels of expressivity (which reflect the three configurations of Triple Space), i.e. the number of coordination patterns supported increase as one moves from the Core API level to Extended (e.g. adds publish/subscribe) and Further Extended APIs (e.g. adds transactionality). Furthermore, the implementation of API operation support differs between components (e.g. for the Transaction Manager, it only makes sense once the Further Extended API is supported). Thus, for the final implementation we support the Further Extended API enabling access to the full potential of the kernel.

The following tables explain the API operations using abstract classes. In API groundings, these classes are mapped to actual classes in a chosen API. Remote groundings such

as SOAP define their own operations for request and response actions, and the API operations are contained within the communication content (e.g. SOAP message) typically as string values of an "operation" parameter.

The definitions of the used abstract classes are:

Triple - atomic data unit equivalent to a RDF statement

Set<Triple> - a set of Triple objects, which forms as a whole a single RDF graph

Template - a generic query

SimpleTemplate - a triple pattern (triple which can contain variables); the simplest query possible

URI - a URI reference to a triplespace

Time - a time measurement to indicate the duration for which a kernel shall wait for the resolution of the query before unblocking the requesting process

Boolean - a boolean value

Core API

Operation	Returns	Description
out(Triple t, URI space)	void	Atomically writes a single triple into the space. The operation makes no guarantee if and when the triple will be available in the space (unordered semantics). The client is immediately free to perform further activities. The client has to provide a resolvable URL which identifies a space.
rd(SingleTemplate t, URI space, Time timeout)	Set<Triple> s	Returns one match of the given template which is a single triple pattern. The match may be a set of triples, e.g. Concise Bounded Description. The operation makes no guarantee as to when the match would be returned to the client. A timeout is provided to give a temporal bound for returning a match. If no match has been found by the timeout period, an empty set is returned. This does not make any statement regarding the existence of a match in the space. No timeout can be specified by providing a null value to the timeout parameter.
rd(SingleTemplate t, Time timeout)	Set<Triple> s	As the rd operation above but no space URI is specified. The system is free to select a match from anywhere in Triple Space where the client has read permissions.

Extended API

Operation	Returns	Description

out(Set<Triple> t, URI space)	void	Atomically writes a set of triples into the space. The operation makes no guarantee if and when the triples will be available in the space (unordered semantics). The client is immediately free to perform further activities. The client has to provide a resolvable URI which identifies a space.
rd(Template t, URI space, Time timeout)	Set<Triple> s	Returns one match of the given template. The match may be a set of triples, e.g. Concise Bounded Description. The operation makes no guarantee as to when the match would be returned to the client.A timeout is provided to give a temporal bound for returning a match. If no match has been found by the timeout period, an empty set is returned. This does not make any statement regarding the existence of a match in the space. No timeout can be specified by providing a null value to the timeout parameter. More expressive templates can be supported by Triple Space implementations than the single triple pattern of the Core API, e.g. SPARQL Queries. The actual set of triples returned will be determined by the definition of the matching rules.
rd(Template t, Time timeout)	Set<Triple> s	As the rd operation above but no space URI is specified. The system is free to select a match from anywhere in Triple Space where the client has read permissions.
rdmultiple(Template t, URI space, Time timeout)	Set<Set-<Triple>> s	As the rd operation above but returns multiple matches of the given template. Each match may be a set of triples, e.g. Concise Bounded Description. There is no completeness guarantee, i.e. the answer given to the client may not contain all matches within the space.
subscribe(Template t, Callback c, URI space)	URI subscription	A subscription is established by providing a template and a callback. When a set of triples matching the template is atomically outed into the specified space, the callback is sent that matched set of triples. The operation returns an URI identifying the subscription.
unsubscribe(URI subscription)	void	This cancels the active subscription with the given URI if it exists.

Further Extended API

Operation	Returns	Description
in(Template t, URI space, Time timeout)	Set<Triple>	As rd, but deletes eventually within the given space the matched triples as determined by the matching rules. It is possible that Triple Space, with the principle of persistent publication, does not need any destructive read functionality.
inmultiple(Template t, URI space, Time timeout)	Set<Set-<Triple>>	As rdmultiple, but deletes eventually multiple matched triples as determined by the matching rules.

createTransaction(String type)	URI transactionID	Type can be “local” or “shared”. The idea is to support both transactions which are local to a client (URI is only known to the client) and are shareable with others (for distributed transactions). Identifiers of shared transactions would be made available to other clients, who can then ’get’ the shared transaction to participate in it (see getTransaction). Transactions may be supported optimistically or pessimistically, the latter would in the case of Triple Space potentially make less guarantees (weak ACIDity).
getTransaction(URI transactionID)	Boolean result	Used to join in a shared transaction with other clients. If true is returned, the client now shares in this transaction once it is begun until it is committed or rolled back.
beginTransaction(URI transactionID)	Boolean result	Begins the transaction. All subsequent interactions by all agents sharing this transaction are handled transactionally, i.e. ’all or nothing’.
commitTransaction(URI transactionID)	Boolean result	commits all interactions made within this transaction in the space.
rollbackTransaction(URI transactionID)	Boolean result	rolls back all interactions made within this transaction in the space.

2.1.2 Management API

Furthermore, a separate API has been developed for operations carried out between kernels or by space administrators, and hence do not belong in the public Triple Space API for normal clients. The *Management API* was derived from the requirements of components in Triple Space where changes had to be effected by administrators outside of the kernel.

2.2 Non-Functional Features

This section describes the security mechanisms and transaction mechanism supported by the prototype impelementation.

2.2.1 Transactions

The transaction support in the context of Triple Space differs from a typical database transaction. As first step we define the transactional support will be valid only in the context of single kernel. Triple space data model guarantee completeness in the scope of single kernel and instantaneous delete after data removal. Hence, the direct approach for the implementation of ACID support seems impractical because the existing data model could not ensure the implementation of reliable database transactions even in ”a perfect world”, where all user inputs are correct, operation executed sequentially and the computer hardware operates in downtime free environment.

We present an use cases which demonstrates a basic transactional need when the client interacts with the Triple Space. Client A would like to check in isolated manner whether newly inserted set of statements (D1), will alter the state of a given space (S1)

Operation	Returns	Description
create(String path, URI space)	URI	Creates a new (sub-)space. The client provides a path and a space and the operation returns a new URI which identifies the newly created subspace. Its use will be controlled by the access control policies; as a rule only the space administrator can create new subspaces in their space and only the kernel administrator can create new subspaces of the root space.
destroy(URI space)	void	This deletes eventually the space identified by the given URI. This also eventually deletes all child spaces of this space.
addMetadata(URI space, Graph g)	void	This operation adds the triples provided in the RDF Graph g to the metadata for the space with the given URI. For this operation to successfully execute, the space given must reside on the kernel being contacted.
getCatalogue(URI space)	Catalogue	returns a listing of the local space structure
setPolicy(URI space, Set<Triple> policy)	void	This operations sets the access policy for the specified space and its subspaces. The policy is given as SAML security assertions.
getPolicy(URI space)	Set<Triple>	Retrieves the access policies for the specified space.
lookup(URI space)	KernelAddress	Returns the kernel that hosts the specified space.

Table 2.4: Management API operations

to new state (S2). The difference between S1 and S2 is the data D1 and D2 (with D2 we indicate the implicitly inferred data). If Client A finds inconsistencies in the information (by definition the space is always passive and does not perform consistencies checks), it must be able to rollback its changes in easy way and preserve the other users from seeing it. The alternative to use inmultiple is most likely not feasible and reliable way to quickly remove the newly inserted information. So, the transactional support in context of Triple Space is used as a tool for the clients to provide easy mechanism for write request abortions. Hence, there is need of shared transactions - when two users have to agree over specific newly inserted content and the resulting implicit knowledge derived by the space.

The Transaction Manager adds transaction support to the Triple Space kernel and controls all components to persist their internal state. It handles the creation of new transactions and coordinates the commit and rollback. After starting a transaction subsequent operations are passed over the API within that transaction. Changes to the storage are handled transactionally within the RDF storage layer, which is no problem as the ORDI framework supports transactions. Every operation, which is executed within a transaction, is logged by the TransactionManager in a "transaction log". To accomplish this the Transaction Manager monitors the operations on the integration space. When an operation is performed transactionally, the Transaction Manager checks whether the transaction id is valid and adds the operation to the transaction log. If the client refers to an invalid transaction identifier an error is raised. As the basic principle, all operations on the RDF storage which take place as an effect of the (kernel internal) processing of one of the operations within the TS-API transaction, have to be executed within a corresponding ORDI transaction. Importantly, in the case of a rollback, all components need to take care of compensation actions (if necessary).

Transaction support is an optional feature that can be enabled by setting the system

property "org.tripcom.transaction.enabled" to TRUE. If disabled, the transaction manager is not required at all. If a client requests transactionality on a kernel where the feature is not available, the TS-API throws a suitable exception.

2.2.2 Security

Triple Space kernels implement the TripCom *security model* defined in deliverable D5.2 [7] and further refined in deliverables D5.3 [4] and D5.4 [3]. This model provides for access control functionalities and permits kernel administrators and triplespace owners to control which API operations can be performed by clients.

Security features can be enabled or disabled on a per-kernel basis, and kernels with different security settings can interoperate with each other. Disabling security means that every operation is allowed on a kernel, except security-related management operations (SET_POLICY and GET_POLICY) which are unavailable. This settings is useful for kernels serving public data or operating in a closed-world scenario where users are known a-priori and access control features are not required.

With security enabled, a client using the TripCom API sends requests to a kernel over a secure TSL channel, authenticating itself with a client X509 *certificate* and possibly providing SAML *attribute assertions* issued by third party *attribute authorities*. TLS is used to provide for data confidentiality and integrity; it is used also for inter-kernel communications, in order to support the *secure forwarding* of client requests. Attribute assertions serve to certify client properties relevant from a security point of view and represent the main information used in the access control process. The certificate, finally, permits to authenticate the client and to prove the possession of the provided attribute assertions.

Access control on the kernel side is regulated by Triple Space *policies*, which are expressed according to a Triple Space Security *ontology* whose final version is reported in appendix in deliverable D5.4 [3]. A security policy consists of *trust rules*, *attribute mapping rules* and *access control rules*. The firsts permit to filter the SAML attributes provided by the client, retaining only the attributes issued by trusted authorities or, *transitively*, by authorities related to them by trust relationships, according to the extended trust and attribute mapping model introduced in deliverable D5.4 [3]. Mapping rules permit to associate *security* roles to clients if certain (trusted) attributes are present. Access control rules, finally, are inspired by XACML and specify which operations can or cannot be performed by clients with certain security roles, according to a role-based access control approach. The central role played by attributes stems from the open-world nature of the Triple Space, and permits to associate security roles and take access control decisions without any need for a previous a-priori knowledge of clients and, to a certain extent and due to transitive trust, of attribute authorities.

Security policies are employed at a kernel and triplespace level. A *space policy* is defined by the space owner using the GET_POLICY and SET_POLICY operations of the Management API; a space policy specifies which operations can be performed on the space and must obey to the constraints of parent space policies, whose access control decisions can prevail (according to the configured policy combining algorithms). A *kernel policy*, on the other hand, is specified in a configuration file by the kernel administrator and regulates the creation (CREATE operation) of new root spaces or distributed spaces on the kernel.

Finally, as part of the Triple Space prototype a couple of tools have been developed to assist users in exploiting the security features of Triple Space. The *policy validator* tool supports space owners to check a security policy for consistency and adherence to the Triple Space Security ontology. An *assertion generator* tool provides for the generation of SAML assertions and, when used with OpenSSL[1] or similar utilities, permits to generate and deploy all the required cryptographic material without any need to access or operate a complex attribute authority.

[1] `http://www.openssl.org/`

3 Prototype Architecture

This chapter provides a brief description of the architecture of the TripCom prototype including a description of all components.

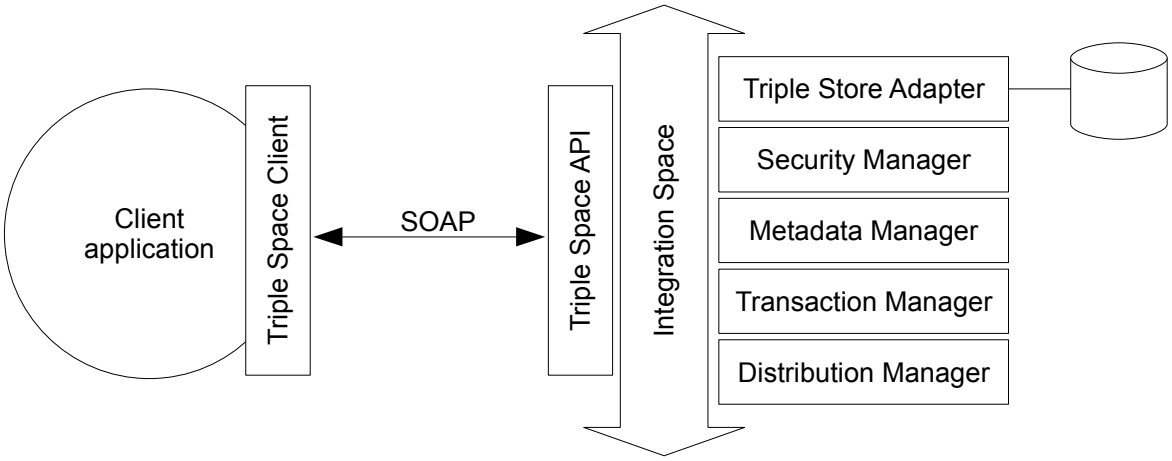

Figure 3.1: The architecture of the TripCom prototype implementation.

The prototype is separated into several components which are integrated via a JavaSpace. This means that the components do not communicate directly with each other. Instead, they comunicate by means of exchanging tuples via a tuple space. For the implementation of the TripCom prototye the JavaSpace implementation Blitz[1] has been choosen. Figure 3.1 depicts the architecture of the tripcom prototype. The left side of the image depicts a client appilication which uses the Triple Space Client component to communicate with the kernel. The kernel is depcited at the right side of the image. As mentioned before, the communication between the components takes place via the integration space. The only component which has a persistent storage is the Triple Store Adapter. See [8] for a detailed description of the components and the integration of the components.

Triple Store Adapter : This component waits for requests to the triple store and uses the appropriate storage API to read and persistently store data. Every other com ponent may use this one to store runtime data (e.g. a components internal state) into persistent storage.

Security Manager : This component verifies that requested operations do not violate the specified security policy. The security manager implements the policy decision point. The policy enforcement point is implemented in the virtual shared memory based bus system.

Metadata Manager : The Metadata Manager is responsible for the implementation of the functionality described by the TripleSpace ontology. It acts as a reasoner to the TSOntology that takes care of validation of ontology instance data across its schema and reasons to find out the information about data as required by users, e.g. calculat ing access statistics for triples, graphs, namedgraphs, logical subspaces etc.

[1] http://www.dancres.org/blitz/

Transaction Manager : This component manages local transactions. The main job of the transaction manager is to coordinate the commit, resp. the rollback of a transaction. To do so it has to communicate with the Distribution Manager and the Triple Store Adapter and coordinate the commit/rollback process. Additionally, this component is responsible for creating transactions.

Distribution Manager : This component is responsible for the implementation of a distributed TripleSpace, e.g. by forwarding queries and requests to kernels that may be able to partially satisfy them and forwarding of write requests to appropriate kernels as part of a semantic clustering of data.

Triple Space API; Triple Space Client : This component is separated into two parts. The Triple Space API is implemented as servlet and accepts SOAP messages which are translated into tuples for the internal integration space. The second part of the component, the Triple Space Client is a library which exports the triple space API methods and can be used to create and send SOAP message to the Triple Space API component.

4 Installation and Configuration Instructions

This chapter explains how the kernel can be compiled, deployed, configured and started. It also lists the applications and libraries the kernel depends on. Furthermore we will show a short example of how a client application can use the prototype.

4.1 Required Applications

This section lists the applications that the prototype depends on, that is the middleware used to integrate the components and the servlet container required to access the Triple Space from a remote host. It provides installation and configuration instructions necessary to properly set up these applications for the prototype.

In order to compile and use the prototype the following software has to be installed:

- Java Development Kit: JDK 1.6[1].
- Maven software development tool: Maven 2.0[2]
- Subversion version control system: Subversion 1.4[3].
- Tomcat: Tomcat 6.0[4]
- Blitz Java Spaces: Blitz 2.0[5]
- Jini: Jini 2.1[6]

Please note that Subversion is only required if one wants to download the latest version from the development repository. All these applications and frameworks can be installed as usual. If required installation instructions can be found on the respective Web sites. The following sections describe all steps necessary to configure those applications after they have been installed using default settings.

4.2 Blitz

The default configuration of Blitz uses network multicasts to find the *reggie* registry. When another registry is found, Blitz will automatically distribute its JavaSpaces entries (if there is another Blitz instance running). Thus, if multiple independent kernel instances shall be hosted within one network, the multicast lookup has to be turned off and singlecast lookup has to be used. Since Blitz uses multicast lookup per default, the following lines have to be added to `config/blitz.config` to tell Blitz where it can find the jini registry (reggie) without multicasts:

[1] `http://Java.sun.com/Javase/downloads/index_jdk6.jsp`
[2] `http://maven.apache.org/`
[3] `http://subversion.tigris.org/`
[4] `http://tomcat.apache.org/tomcat-6.0-doc/index.html`
[5] `http://www.dancres.org/blitz/`
[6] `http://www.jini.org/`

```
...
import net.jini.core.discovery.LookupLocator;
...
org.dancres.blitz {
        ...
        initialLocators =  new LookupLocator[]
                               {new LookupLocator("jini://localhost")};
        ...
}
```

Furthermore the Jini transaction manager (*mahalo*) needs to know where it has to register at startup. Therefore the following lines have to be added to `conf/mahalo.conf`.

```
....
import net.jini.core.discovery.LookupLocator;

com.sun.jini.mahalo{
        ...
        initialLookupLocators = new LookupLocator[]
                                    {new LookupLocator("jini://localhost")};
}
```

In order to disable multicasts the jini registry has to be configured to not use any network interface for broadcasting. The following lines have to be added to `conf/reggie.conf`.

```
...
import java.net.NetworkInterface;

com.sun.jini.reggie{
        ...
        multicastInterfaces = new NetworkInterface[]{};
}
net.jini.discovery.LookupDiscovery{
        multicastInterfaces = new NetworkInterface[]{};
}
```

Blitz has its own monitoring tool which is called *dashboard*. The dashboard uses multicast lookup to find the JavaSpace instance. In order to use the dashboard without broadcast lookup, the hostname and the port have to be passed as arguments in the dashboard start script `dashboard.sh`:

```
...
$JAVA_HOME/bin/Java $POLICY -cp $CP \
  org.dancres.blitz.tools.dash.StartDashBoard \
  localhost:4160 $SPACE_NAME
```

4.2.1 Performance Optimization (optional)

Per default, Blitz persists all written entries. This is not necessary because Blitz is only used for inter-component communication. The persistence model of Blitz can be

configured by modifying the "storageModel" variable in `conf/blitz.config`. This variable has to be set to `storageModel = new Transient();`. Additionally, the number of threads which are used to read entries from the space can be increased. This can be configured by modifying the `maxTaskThreads` variable. For example: `maxTaskThreads = 60;`. The above variables are set in the default configuration of Blitz. Therefore, these variables should not be added to the configuration. Instead, the existing variables should be altered.

4.2.2 Starting Blitz

In the Blitz start script (`blitz.sh`) the home directory of the Java virtual machine (`JAVA_HOME`) has to be set. After starting Blitz it creates log and database files in the `log` directory. In order to have a clean Blitz instance after each restart this directory can be removed before starting Blitz. This can be done by adding a command which deletes the content of the `log` directory to the Blitz start script (for instance on Linux `rm -rf log/*`).

4.3 Tomcat

The API operations are implemented as Web Services using the Apache Axis [7]. Tomcat is used to deploy these Web Services. After the default installation one has to do the following steps:

1. Compile the API web application using Maven with the following command (see also Section4.6.2):

   ```
   mvn clean package assembly:assembly war:war -Dmaven.test.skip
   ```

   command with the above mention maven command

2. Copy the generated WAR file to the `axis` subdirectory in the tomcat installation directory.

3. Start Tomcat.

4.4 Using TLS (optional)

NOTICE: The tripcom TLS setup does not work with tomcat 6. Please download the latest version of tomcat 5 if you want to use TLS.

In order to to be able to contact a kernel via Transport Layer Security (TLS), several additional steps have to taken. The following pathnames will be abbrevated like this:

- `TOMCAT_HOME`– tomcat's directory
- `JAVA_HOME` – the place of the java JVM

[7] `http://ws.apache.org/axis2/`

4.4.1 Configuring tomcat

adjusting the `server.xml` file

Add the following to tomcat's `server.xml` configuration file in order to have it listen for TLS connections on Port 8443:

```
<Connector port="8443" maxHttpHeaderSize="8192"
               maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
               enableLookups="true" disableUploadTimeout="true"
               acceptCount="100" scheme="https" secure="true"
               clientAuth="want" sslProtocol="TLS"
               keystoreFile="$TOMCAT_HOME/SERVER.KEYSTORE" keyAlias="KEYALIAS"
               keystorePass="KEYPASS"
               truststoreFile="$TOMCAT_HOME/server.dummy_truststore"
               truststorePass="changeit" truststoreAlgorithm="tripcom" />
```

Pathes and entries with values all in captials (keystorefile, keyalias, keystorepass) must be replaced accordingly. The server keystore must contain a single certificate.

Setting up a dummy truststore for tomcat

Although tomcat will be modified to accept all client certificates (see below), tomcat also needs a truststore file, which usually contains certificates it accepts. Anyone will do, but you can download a dummy one from `http://tripcom.sf.net/ssl/server.dummy_truststore` and place it in `TOMCAT_HOME`. The above entry in the `server.xml` already has the appropriate entries for this.

4.4.2 Installing the tripcom security provider

In order to have tomcat trust all certificates, one needs a custom java security provider. Download `http://tripcom.sf.net/ssl/tripcomprovider.jar` and place it in `JAVA_HOME/jre/lib/` Next, edit the `JAVA_HOME/jre/lib/security/java.security` file to make it aware of this provider. As of java 1.5, five security providers are listed in the this file. The last one look should like this:

```
security.provider.6=com.sun.security.sasl.Provider
```

Add the following line below it (and adjust the number as necessary):

```
security.provider.7=org.tripcom.external.TripcomProvider
```

If one has a client certifiacte at hand, one can test the setup on a local machine by opening `https:localhost:8443axis`. After accepting the server's certificate, you should be prompted for your client certificate. Both chosing any certificate or none at all should lead you to the apache axis webpage.

4.5 Required Libraries

In this section we list all 3^{rd} party dependencies required to build and use the prototype. All of these dependencies are publicly available and free. Most of them are downloaded automatically by Maven from public repositories. If a particular dependency is not available in the repositories, it can be manually downloaded and installed to the maven repository. The files can easily be found using standard Internet search engines such as Google.

```
org.tripcom:main:jar:1.0-SNAPSHOT
 +- junit:junit:jar:4.4:test (scope not updated to compile)
 +- org.tripcom.distribution:DistributionManager:jar:1.0-SNAPSHOT:compile
 |  +- com.hp.hpl.jena:arq:jar:2.4:compile
 |  |  +- org.apache.lucene:lucene-core:jar:2.3.1:compile
 |  |  +- stax:stax-api:jar:1.0:compile
 |  |  +- woodstox:wstx-asl:jar:3.0.0:compile
 |  |  \- com.hp.hpl.jena:json-jena:jar:1.0:compile
 |  +- com.hp.hpl.jena:jena:jar:2.5.4:compile
 |  |  +- com.hp.hpl.jena:arq-extra:jar:2.1:compile
 |  |  |  \- com.hp.hpl.jena:jenatest:jar:2.5.4:compile
 |  |  +- com.hp.hpl.jena:iri:jar:0.5:compile
 |  |  +- antlr:antlr:jar:2.7.5:compile
 |  |  +- commons-logging:commons-logging-api:jar:1.1:compile
 |  |  +- com.hp.hpl.jena:concurrent-jena:jar:1.3.2:compile
 |  |  +- com.ibm.icu:icu4j:jar:3.4.4:compile
 |  |  +- org.codehaus.woodstox:wstx-asl:jar:3.0.0:compile
 |  |  +- xerces:xercesImpl:jar:2.7.1:compile
 |  |  \- xerces:xmlParserAPIs:jar:2.0.2:compile
 |  +- esper:esper:jar:1.11.0:compile
 |  |  +- cglib:cglib-full:jar:2.0.2:compile
 |  |  \- mysql:mysql-connector-java:jar:3.1.14:compile
 |  +- org.openrdf:openrdf-sesame-onejar:jar:2.0:compile
 |  +- org.apache.axis:axis-jaxrpc:jar:1.4:compile
 |  +- javax.activation:activation:jar:1.1:compile
 |  +- javax.mail:mail:jar:1.4:compile
 |  +- hsqldb:hsqldb:jar:1.8.0.7:compile
 |  +- xmlbeans:xmlbeans:jar:2.0-dev-2:compile
 |  +- org.slf4j:slf4j-api:jar:1.5.0-M0:compile
 |  +- commons-logging:commons-logging:jar:1.1:compile
 |  |  +- logkit:logkit:jar:1.0.1:compile
 |  |  +- avalon-framework:avalon-framework:jar:4.1.3:compile
 |  |  \- javax.servlet:servlet-api:jar:2.3:compile
 |  +- org.restlet:org.restlet:jar:1.0.1:compile
 |  +- proxool:proxool:jar:0.8.3:compile
 |  +- h2:h2:jar:1.0:compile
 |  +- NBench:NBench:jar:1.0:compile
 |  +- proxool-cglib:proxool-cglib:jar:1.0:compile
 |  +- p-grid:p-grid:jar:3.2.0_822:compile
 |  +- getopt:getopt:jar:1.0.8:compile
 |  +- org.tripcom.integration:integration:jar:1.0-SNAPSHOT:compile
```

```
|  |  +- net.jini:jini-core:jar:2.1:compile
|  |  +- net.jini:jini-ext:jar:2.1:compile
|  |  +- net.jini:jsk-lib:jar:2.1:compile
|  |  +- backport-util-concurrent:backport-util-concurrent:jar:3.1:compile
|  |  \- berkeleydb:je:jar:3.2.44:compile
|  \- org.tripcom.tsclient:tsclient:jar:1.0-SNAPSHOT:compile
|     \- at.telekom:axistools:jar:1.0:compile
+- org.tripcom.metadata:metadata:jar:1.0-SNAPSHOT:compile
+- org.tripcom.security:sm:jar:1.0-SNAPSHOT:compile
|  +- org.springframework:spring-context:jar:2.5.1:compile
|  |  +- aopalliance:aopalliance:jar:1.0:compile
|  |  +- org.springframework:spring-beans:jar:2.5.1:compile
|  |  \- org.springframework:spring-core:jar:2.5.1:compile
|  +- org.opensaml:opensaml:jar:2.2.2:compile
|  |  +- commons-collections:commons-collections:jar:3.1:compile
|  |  +- commons-lang:commons-lang:jar:2.1:compile
|  |  +- jargs:jargs:jar:1.0:compile
|  |  +- velocity:velocity:jar:1.5:compile
|  |  +- org.apache.xerces:xml-apis:jar:2.9.1:runtime
|  |  +- org.apache.xerces:xercesImpl:jar:2.9.1:runtime
|  |  +- org.apache.xerces:resolver:jar:2.9.1:runtime
|  |  +- org.apache.xerces:serializer:jar:2.9.1:runtime
|  |  \- org.apache.xalan:xalan:jar:2.7.1:runtime
|  +- org.opensaml:openws:jar:1.2.1:compile
|  |  +- commons-codec:commons-codec:jar:1.3:compile
|  |  \- commons-httpclient:commons-httpclient:jar:3.1:compile
|  \- org.opensaml:xmltooling:jar:1.1.1:compile
|     +- org.slf4j:jcl-over-slf4j:jar:1.5.5:compile
|     +- org.slf4j:log4j-over-slf4j:jar:1.5.5:compile
|     +- joda-time:joda-time:jar:1.5.2:compile
|     +- org.bouncycastle:bcprov-ext-jdk15:jar:1.40:compile
|     +- org.apache.santuario:xmlsec:jar:1.4.2:compile
|     \- org.apache.commons.ssl:not-yet-commons-ssl:jar:0.3.9:compile
+- org.tripcom.security:smout:jar:1.0-SNAPSHOT:compile
+- org.tripcom.transaction:transaction:jar:1.0-SNAPSHOT:compile
+- org.tripcom.tsadapter:tsadapter:jar:1.0-SNAPSHOT:compile
|  +- ordi:ordi-model:jar:0.5-SNAPSHOT:compile
|  |  +- org.openrdf.sesame:sesame-model:jar:2.1:compile
|  |  |  +- info.aduna.commons:aduna-commons-collections:jar:2.0:compile
|  |  |  |  \- info.aduna.commons:aduna-commons-concurrent:jar:2.0:compile
|  |  |  \- info.aduna.commons:aduna-commons-iteration:jar:2.0:compile
|  |  +- org.openrdf.sesame:sesame-query:jar:2.1:compile
|  |  |  \- org.openrdf.sesame:sesame-rio-api:jar:2.1:compile
|  |  +- org.openrdf.sesame:sesame-queryalgebra-model:jar:2.1:compile
|  |  +- org.openrdf.sesame:sesame-queryalgebra-evaluation:jar:2.1:compile
|  |  |  \- info.aduna.commons:aduna-commons-lang:jar:2.0:compile
|  |  +- org.openrdf.sesame:sesame-queryparser-api:jar:2.1:compile
|  |  +- info.aduna:aduna-collections:jar:1.4:compile
```

```
|  |  |  \- info.aduna:aduna-concurrent:jar:1.4:compile
|  |  +- info.aduna:aduna-iteration:jar:1.5:compile
|  |  \- info.aduna:aduna-net:jar:1.5:compile
|  |     \- info.aduna:aduna-text:jar:1.3:compile
|  \- ordi:ordi-trree-adapter:jar:3.0alpha4-SNAPSHOT:compile
|     +- ordi:ordi-test:jar:0.5-SNAPSHOT:compile
|     |  \- org.openrdf.sesame:sesame-rio-rdfxml:jar:2.1:compile
|     |     \- info.aduna.commons:aduna-commons-xml:jar:2.0:compile
|     +- trree:trree:jar:3.0-alpha5:compile
|     \- org.openrdf.sesame:sesame-queryparser-sparql:jar:2.1:compile
|        +- info.aduna.commons:aduna-commons-net:jar:2.0:compile
|        \- info.aduna.commons:aduna-commons-text:jar:2.0:compile
+- org.tripcom.tsapi:tsapi:jar:1.0-SNAPSHOT:compile
|  +- jini:jini-core:jar:2.1:compile
|  +- jini:jini-ext:jar:2.1:compile
|  +- org.slf4j:slf4j-log4j12:jar:1.5.0-M0:compile
|  \- org.apache.tomcat:servlet-api:jar:6.0.13:compile
+- org.dancres:blitz:jar:2.0-rc4:compile
+- org.dancres:blitz-dl:jar:2.0-rc4:compile
+- org.mortbay.jetty:jetty:jar:6.1.10:compile
+- org.mortbay.jetty:jetty-util:jar:6.1.10:compile
+- org.mortbay.jetty:servlet-api-2.5:jar:6.1.10:compile
+- axis:axis:jar:1.4:compile
|  +- org.apache.axis:axis-saaj:jar:1.4:compile
|  +- axis:axis-wsdl4j:jar:1.5.1:runtime
|  \- commons-discovery:commons-discovery:jar:0.2:runtime
\- log4j:log4j:jar:1.2.14:compile
```

4.6 Building, Configuring and Starting the Protoype

This section describes how the prototype is compiled, configured and started.

4.6.1 Obtaining the source

First, the source code has to be checked out from the sourceforge Subversion repository[8]. This can be done using the Subversion command line tool or via a GUI extension. From the command line one has to type the following line to download the latest version of the prototype.

```
svn co https://tripcom.svn.sourceforge.net/svnroot/tripcom/trunk
```

This command copies all tripcom related source code to the local working directory. A directory called `trunk` is created where the files are stored.

4.6.2 Compiling the source

All kernel components can be build with the following command (It has to be executed in the directory `.../trunk/`):

[8] `http://sourceforge.net/svn/?group_id=169344`

```
mvn clean package assembly:assembly war:war -Dmaven.test.skip
```

By default, Maven executes the test cases of all components after compiling. To skip the execution of these tests `-Dmaven.test.skip` is added.

After a successful compilation Maven creates a zip (`tripcom-${version}-bin.zip`) archive in the directory `target`. The zip archive contains the following files and directories:
/libs all third party libraries,
/bin the kernel components,
/conf configuration files,
Furthermore it contains several start scripts **start.sh, start.bat, etc.** for Unix and Windows (and the bash scripts can used on MacOS too).

4.6.3 Kernel configuration

After unpacking the zip archive, the TripCom kernel can be configured using the configuration files in the `conf` directory, e.g. `distribution-manager.properties` contains configuration settings for Distribution Manager, `security-manager.properties` configures the Security Manager, etc. The various configurable settings are documented inside the configuration scripts. A single kernel can be started using the default values. To connect a kernel to a network of kernels using P-Grid, that kernel has to know the IP of one those other kernels. Therefore the value of `org.tripcom.distribution.controller.BOOTSTRAP_HOST_ADDRESS` in the Distribution Manager settings has to be set to the IP address of that kernel. For a single kernel or the first kernel of a set of kernels, this property can be set to `localhost` (which is the default). An other important setting is the address of the kernel, i.e. its host name (or IP) and its port. This has to be set in the relevant start script by changing the value of `KERNEL_ADDRESS`. By default this property is set to `localhost:8080` which works for a single kernel. However, when multiple kernels are communicating with each other they sometimes have to exchange their addresses so that one kernel knows how the reach the other kernel. In this case the `KERNEL_ADDRESS` must be a resolvable host name or IP that can be used by other kernels to connect to that kernels.

4.6.4 Starting the kernel

In order to start the kernel the provided start scripts can be used. The kernel can be started with `start.sh` on Unix and with `start.bat` on Windows machines, respectively. When starting the kernel Blitz must already be running. In order to access the kernel using the API Web Service Tomcat has be started too.
Furthermore there are start scripts that start all components, an embedded Jetty server[9] and (optionally) also an (embedded or stand-alone) Blitz instance. This start scripts are much more convenient, but some security features will not work when using jetty, which is why they are only recommended for testing purposes.

[9] `http://www.mortbay.org/jetty/`

```
import org.openrdf.model.*;
import org.tripcom.api.ws.client.*;

public class Example {

 public static void main(String args[]){
  // The host name of TS Kernel
  String host = "localhost";
  // The Port of the TS Kernel
  int port = 8080;
  // target Space URL
  String space = "tsc://" + host + ":" + port + "/someRootSpace";
  TSClient myClient = new TSClient("http", InetAddress.getByName(host),
    port, CERTIFICATE);
  // create the space
  myClient.create(space);
  // the tripple which is written into the space in xml format.
  String rdfxmlString = "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n"
    + "<rdf:RDF xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\"\n"
    + "xmlns:dc=\"http://purl.org/dc/elements/1.1/\">\n"
    + "<rdf:Description rdf:about="
    + "\"http://de.wikipedia.org/wiki/Resource_Description_Framework\">\n"
    + "<dc:title>Resource Description Framework</dc:title>\n"
    + "<dc:publisher>Wikipedia - Die freie Enzyklopdie</dc:publisher>\n"
    + "</rdf:Description>\n" + "</rdf:RDF>";
  // write the triple
  myClient.out(rdfxmlString, space);
  String query = "SELECT ?s ?p ?o WHERE { ?s ?p ?o . } ";
  String s = myClient.rd(query, space, 3000);
  System.out.println(s);
 }
}
```

Figure 4.1: Triple space example

4.7 Example client code

This section presents the source code of a simple client application. It will be shown how a client connects to a kernel and uses the API methods to OUT to and RD from a triple space:

In the example source code depicted in Figure 4.1 at first an instance of the `TSClient` class is created. Besides the address of the kernel to connect to, the constructor requires a string which contains the security information of the client. In this example the variable `CERTIFICATE` contains the clients certificate. Afterwards a root space named `tsc://localhost:8080/someRootSpace` is created. Afterwards an RDF triple in XML format is created and then written into the root space. Finally, the applications retrieves the triple again by using the RD operation and a simple query and the RD operation. The read operation is issued at the space the triple was OUTed too. The timeout of 3000 milliseconds means that the kernel has 3 seconds to return the result.

4.8 License information

The kernel and all tests, tools, scripts, etc. are released under the GNU Lesser General Public License which is available on `http://www.gnu.org/licenses/lgpl.html`. A copy of the license is also included in the root directory of the kernel source code and all source files contain license and copy rights information in the class header. The 3rd party libraries libraries used are available under various different libraries, but all of them are available for free and compatible with the LGPL.

5 EAI Use Case

The EAI use case consists on two different applications, one related to the marketplace implementation to offer content licenses and purchase them using an auction interaction, and the customer side of the aforementioned marketplace. These applications are covered in detail in deliverables [5] and [6] respectively, where the motivation, design of the code and the motivation for the Triple Space application are widely discussed. The code uploaded in the sourceforge is documented to be compiled and used.

6 eHealth Use Case

The eHealth scenario implemented in TripCom aims at demonstrating the TS capabilities to work as semantic middleware for supporting a Patient Summary infrastructure at European Level. Such a scenario, named the European Patient Summary scenario, demands strong requirements in terms of scalability since the supporting infrastructure has to be able to manage the patient summaries of each of the over 500 million European citizens.

The EPS space, intended as a top-level space, can be configured in different ways according to the needs of the participating eHealth authorities. One of the possible configurations is described in [1], with the EPS space subdivided into many subspaces following the natural hierarchical structure of health authorities in Europe, as shown in figure 6.1. Other types of configurations are described in [2], where the EPS space is used to integrate existing country-wide eHealth systems. Although the spaces structures inside a single patient summary may vary, keeping the subdivision of a patient summary in three different subspaces (demographics, PS body and private data) guarantees the possibility of retrieving the summary information about a patient from any of the countries participating to the EPS infrastructure accordingly to the different security policies applied to the subspaces of the desired patient summary.

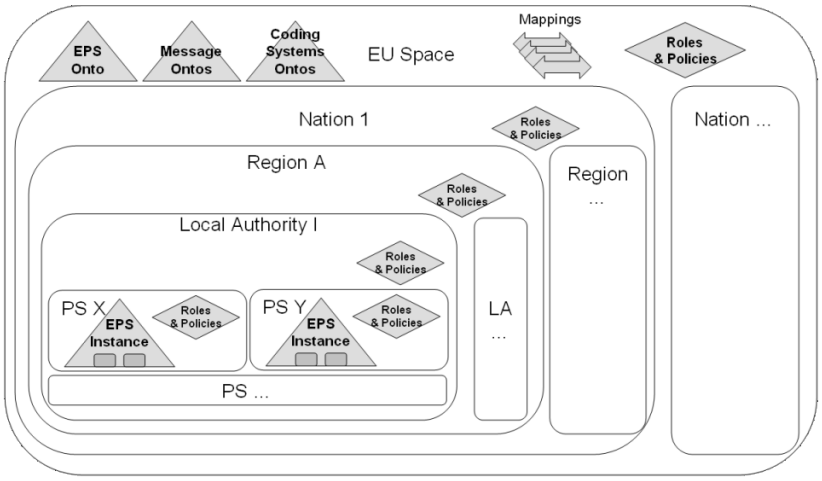

Figure 6.1: Spaces organization from the D8B.2 scenario

6.1 The EPS web application

The EPS web application is a sample application provided in order to test the functionalities of the Triple Space related to supporting the shared care path scenario described in [1] and refined in [2]. It allows a doctor, once correctly logged in, to access the patient summaries hosted on the TS infrastructure according to the security policies defined for the spaces containing them. The authorized doctor can then read and write data of that patient summary, and can also subscribe to changes to summary data of a patient.

Building. Building the EPS webapp is achieved using the "war" task of the provided Ant build file. The task automatically generates a WAR file in the same directory where the build file is located.

Configuration. The web application is designed for demo purposes, and it tries to connect to "london-city.gb.eps.eu" kernel and to "bolzano.altoadige.eps.eu", therefore at least one of these two kernel identifiers must be available.

Installation. Once completed the build step and configured, the WAR file can be deployed in Tomcat as a normal web application.

Deploying EPS spaces and data. For a sample scenario of a triplespace configured as the EPS, you can use the data generation tool and the deployer described in D6.5b, or otherwise deploy the data used for demo purposes that is located in the "data" directory of the source code. Deploying the data used for the demonstration can be achieved by using the "deploy" task of the provided Ant task file.

Usage. In order to access the application, the user must log in. Two users, as also described in [1] and [2], are defined:

- Dr. Georgina Grey: an English doctor, who uses the assertions provided by two English Identity Providers: "UK National Registry of Practitioners" and "London City Health Authority". The former can assert that Dr. Grey is a General Practitioner, while the latter can assert that she is the Responsible General Practitioner for Mr. Corrs. Her login is "georgina grey" and her password is `g12grey`.
- Dr. Paola Pirovano: uses the assertions issued by the "Ordine nazionale dei medici", which identify her as an Italian Periodontist. Her login is "paola pirovano" and her password is `pirovano314`.

Once logged in, the doctor must choose the patient summary by providing one of his identifiers. When using demo data, Mr. Christian Corrs' patient summary can be retrieved by using the passport number "UK-274856 S".

When the patient has been chosen and its patient summary has been successfully retrieved, a set of pages will be available, showing the patient's data and allowing adding new information:

- Demographics
- Social history
- Alerts
- Problems
- Medications
- Encounters
- Immunizations
- People
- Organizations

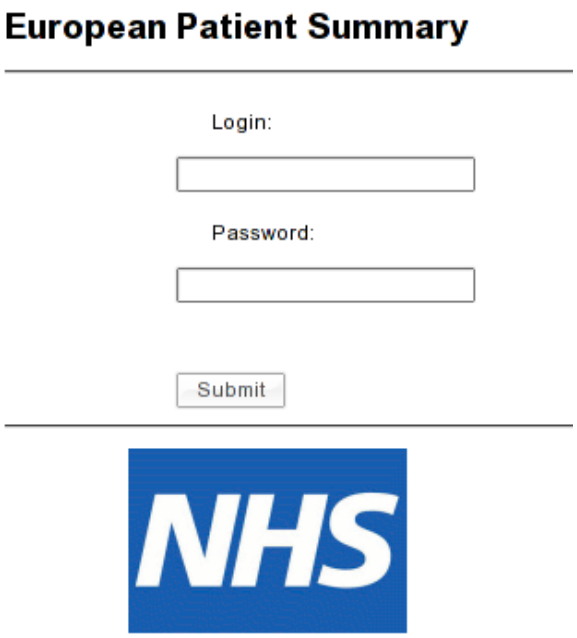

Figure 6.2: EPS webapp: login screen

Figure 6.3: EPS webapp: patient's demographics

These pages reflect the structure of the EPS ontology, and they always contain a view of the data plus a form to insert new information. If the "Keep private" check box is checked, then the data will be added to the restricted access space of the patient summary.

The "People" and "Organizations" pages (see fig. 6.6 report information about the actors that have submitted content to the patient summary. Those pages are intended to be read-only, as those information are automatically sent each time a new entry is added to the patient summary

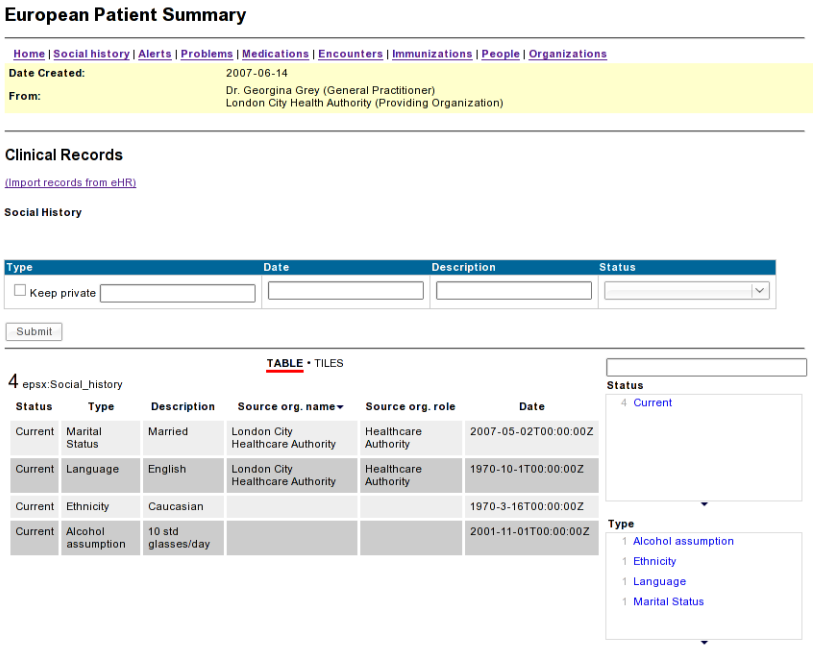

Figure 6.4: EPS webapp: patient's social history

Figure 6.5: EPS webapp: patient's medications

License. The EPS web application, as well as the rest of the TripCom infrastructure, is released under the GNU Lesser General Public License version 2.1.

Dependencies. The following libraries, required to compile and run the EPS web application, are included in the standard distribution:

- Jena 2.5.7 [1]

[1] `http://jena.sf.net`

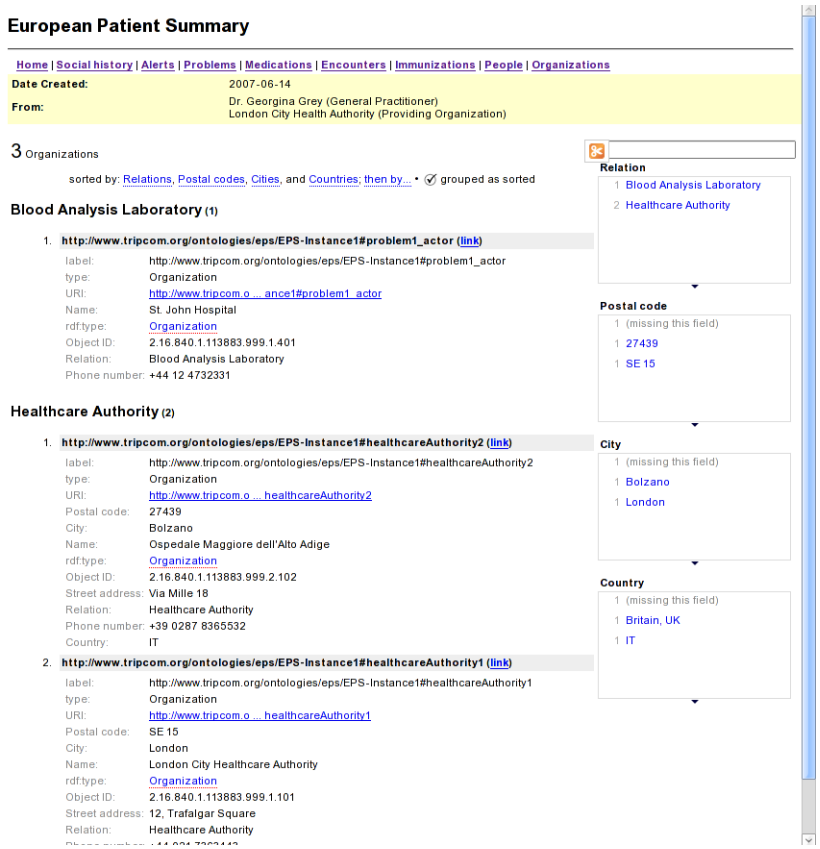

Figure 6.6: EPS webapp: information about the organization that put data inside a patient summary

- Axis 1.4 [2]
- log4j 1.2.14 [3]
- integration library from official TripCom build
- tsclient library from official TripCom build

[2] `http://ws.apache.org/axis/`
[3] `http://logging.apache.org/log4j/1.2/`

References

[1] Carenini, A. and Cerizza, D. and Krummenacher, R. and Foxvog, D. and Martinez, J. and Momtchev, V. and Pérez Crespo, N. Prototype of TripCom application for sharing health data among healthcare organizations. Technical report, TripCom Project Deliverable D8b.2, 2008.

[2] Carenini, A. and Krummenacher, R. and Martinez, J. and De Francisco, D. Assessment of the developed solution with regards to the detected indicator. Technical report, TripCom Project Deliverable D8b.3, 2008.

[3] Davide Cerri, Jacek Kopecký, Kia Teymourian, Michael Lafite, Germán Toro del Valle, and Francesco Corcoglioniti. Final prototype of the security and trust infrastructure. Technical report, TripCom Project Deliverable D5.4, 2009.

[4] Davide Cerri, Hans Moritsch, Jacek Kopecký, Christian Schreiber, and Francesco Corcoglioniti. Early prototype of the security and trust infrastructure. Technical report, TripCom Project Deliverable D5.3, 2008.

[5] David de Francisco Marcos, Guillermo López Reyes, and Germán Toro del Valle. Eai prototype application. Deliverable, TripCom, September 2008.

[6] David de Francisco Marcos, Guillermo López Reyes, and Germán Toro del Valle. Eai validation and recommendations. Deliverable, TripCom, March 2009.

[7] Alessandro Ghioni, Davide Cerri, Jacek Kopecký, Gerson Joskowicz, Lyndon Nixon, Dario Cerizza, Noelia Pérez Crespo, and Francesco Corcoglioniti. Definition of security and trust support model for the reference architecture. Technical report, TripCom Project Deliverable D5.2, 2007.

[8] L.J.B. Nixon, D. Martin, D. Wutke, M. Murth, E. Simperl, R. Krummenacher, B. Sapkota, Z. Zhou, H. Moritsch, C. Schreiber, O. Shafiq, G. Toro del Valle, D. Cerri, and V. Momtchev. Platform API Specification for Interaction Between All Components. TripCom Deliverable D6.3, March 2008.