Peer To Peer Context Data Sharing Through Tuple Spaces

Andrea Detti
University of Rome Tor Vergata,
Electric Engineering Department,
Via del Politecnico 1, Rome, Italy
andrea.detti@uniroma2.it

Pierpaolo Loreti
"Consorzio Nazionale Interuniversitario per le Telecomunicazioni" - UdR University of Rome Tor Vergata,
Via del Politecnico 1, Rome, Italy
pierpaolo.loreti@uniroma2.it

Abstract: **A requirement of pervasive computing systems is context data sharing among software components that collaborate in providing services to users. The solution to this requirement becomes more challenging when the communication scenario is peer-to-peer, i.e. the context data are distributed among components, rather than be held by a centralized server. This paper faces the context data sharing issue proposing a software architecture that realizes a distributed context space by a tuple space. Moreover, the context data are represented using an ontology, described in the Web Ontology Language, thus the system can be seamless integrated with the Semantic Web technologies.**

1. INTRODUCTION

In the pervasive computing vision the environment is fulfilled of software components embedded in common objects (e.g.) enabling them to provide mobile users with context-aware services. As a consequence, the system architecture has to support the seamless integration and cooperation among devices and software components. The integration is not only required from the communication point of view, but it is also necessary in terms of knowledge, i.e. the software components have to share their information about what is going on in the environment [1]. The whole set of environmental information forms the so-called “context”.

Context information sharing relies on a software architecture able to distribute the context data. Moreover, data need to be described in a uniform machine understandable form and a candidate solution is the definition of a common ontology [1, 2]. The ontology unambiguously describes the context data as “concepts” on which software components can reason.

This paper faces the context sharing issue, proposing a peer-to-peer software architecture based on tuple spaces managed by the LIME middleware [3], and a new defined interface (named LIME/OWL Interface) that maps semantic concepts formalized in Ontology Web Language (OWL), [4], on LIME tuples. OWL is a W3C standard originally designed to develop the Semantic Web vision in which the published information has an explicit meaning by the definition of ontologies, making easier for machines to automatically process and integrate available information in order to perform useful reasoning tasks.

We envisage that the peer-to-peer approach, supported by underlying wireless technologies able to realize spontaneous (or ad-hoc) networking, results as a key for fast system deployment, avoiding the need of a provisional infrastructure. In fact, each software component is directly responsible for the management of its context data without the intermediation of a central server. In such dynamic mobile environment, LIME middleware appears as a suitable candidate for data sharing. Finally, the selection of OWL as description language enables the system to be integrated with the Semantic Web technologies.

The rest of the paper is organized as follow: Section 2 describes the peer-to-peer context sharing architecture and the LIME middleware philosophy; Section 4 describes the LIME/OWL interface. Section 4 presents the related works and finally conclusions are drawn.

2. PEER TO PEER CONTEXT SHARING

We refer to the scenario reported in ***Figure 1***, in which a set of software components are running inside some hosting devices that are connected each other via wireless and wired network technologies. Software components write and read (i.e., share) contextual information in terms of OWL concepts forming a logical “Context Space”. The sharing is supported by LIME facilities and the LIME/OWL Interface provides the mapping rules between concept and tuples.

In the paragraph we briefly describe the LIME middleware. Then, in the next Section, we discuss the LIME/OWL Interface and common operation such as putting out and retrieve context data.

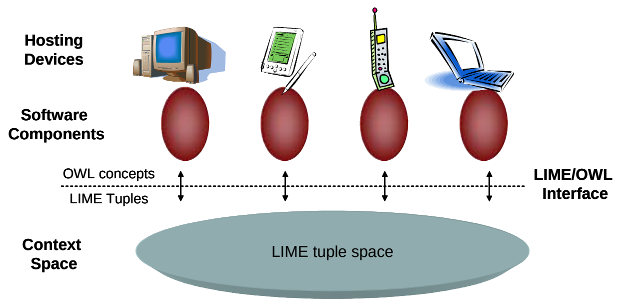


Figure 1 – Reference scenario

2.1. Lime

LIME is a Java-based middleware specifically targeted toward the complexities of the ad-hoc mobile environments, [3]. LIME supports the application design in environments where a physical or logical mobility cause constant changes in the availability of resources (both data and computational elements).

LIME adapts to a mobile environment the notion of Linda where processes communicate by writing, reading, and removing data from a tuple space that is assumed to be persistent and globally shared among all processes. In LIME the tuple space content is distributed across multiple mobile components. When components are within range (i.e., mobile agents are on the same host or communication is available between mobile hosts that contain agents), the contents of the tuple spaces held by the individual mobile components are transiently and transparently shared, forming a federated tuple space. The content accessible through such virtual tuple space, made up of the concrete tuple spaces contributed by each component, changes from time to time according to the current connectivity pattern. Moreover, the fully decentralized architecture increases the system scalability [5].

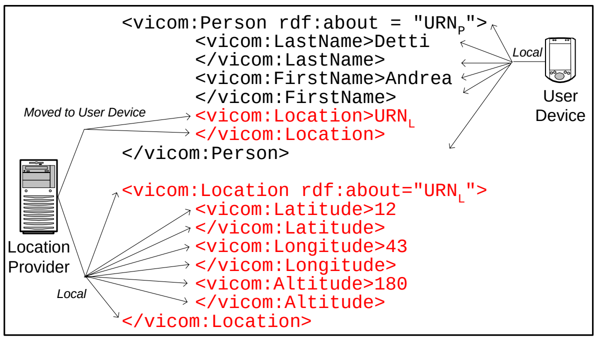
LIME also introduces the notions of tuple location, for querying a partition of the federated tuple space, and of reactive programming, for allowing actions to be performed with varying degrees of atomicity upon insertion of a tuple.

3. LIME/OWL INTERFACE

The LIME/OWL Interface defines rules for mapping the OWL concepts on tuples. Actually, the OWL concepts are represented in RDF document written in XML; as a consequence the LIME/OWL Interface defines the way to publish in a distributed way XML statements consistent with the OWL semantics.

Let us use an example to explain this approach. ***Figure 2*** reports a simplified XML statement representing two OWL individuals belongs to the `Person` and `Location` OWL classes defined in our ontology. These classes are used to describe some user characteristics as profile and location. Each class is uniquely identified by an URI in the form of a URN; `URN`$_P$ identifies the specific `Person` class while `URN`$_L$ identifies the `Location` one. `Person` class is composed by two DataType Properties (DTP): `LastName` and `FirstName`; and by one Object Property (OP): `Location`, which acts as a pointer to the relevant `Location` class. `Location` class is formed by three DTPs: `Latitude`, `Longitude` and `Altitude`.

Referring to ***Figure 2***, the profile information (`LastName` and `FirstName`) is produced by the user device, while the location information is produced by an external location provider. So, the "black" XML TAGs are output by the user device, while the "red" XML TAGs by an external location provider device.



```
<vicom:Person rdf:about = "URN_P">
     <vicom:LastName>Detti
     </vicom:LastName>
     <vicom:FirstName>Andrea
     </vicom:FirstName>
     <vicom:Location>URN_L
     </vicom:Location>
</vicom:Person>

<vicom:Location rdf:about="URN_L">
     <vicom:Latitude>12
     </vicom:Latitude>
     <vicom:Longitude>43
     </vicom:Longitude>
     <vicom:Altitude>180
     </vicom:Altitude>
</vicom:Location>
```

Figure 2 – Example of XML statement

3.1. Dynamic XML Statements over tuples

We define three tuple formats that maps the XML TAGs associated with the Class, DTP and OP OWL concepts. Moreover, these tuple formats contain information that enables to recreate the XML hierarchy. The benchmark philosophy that we adopt in our approach is the following: "the XML over tuples mapping rules must enable the reconstruction of the entire XML statement through a succession of Lime queries on the tuple space". So doing, unlike centralized approach, we "crumble" the XML statement representing the context space on distributed Lime tuples.

The generic tuple format contains two parts: header and context data. The tuple header is a common control part mainly holding the XML hierarchy; while the context data part is specific for each type of tuple and is directly related to OWL concept (Class, DTP or OP) that it represents.

The following sections go into detail about the header and context data parts. However, a preliminary discussion on the concept naming approach is need.

3.2. Concept naming

As it will be clearer in the following, in order to manage the XML hierarchy and to aid the use of unicast queries instead of broadcast one, we need to uniquely identify a concept (e.g., an XML TAG) that is contained in a tuple within the context space. With this aim, we have chosen the URN approach. The concept URN maintains all the information needed to refer the concept in a network environment. The URN contains three parts: i) `host_id`; ii) `agent_id`; iii) `concept_id`.

`host_id` is the network reference, i.e. the IP address and the port number of the Lime server, which host the Lime agent generating the concept. The agent and the concept are respectively identified by `agent_id` and `concept_id`; as the former is unique within the same Lime server, as the latter is for the concepts published by an agent.

It is worth to mention that Lime already provides a unique tuple numbering scheme and recalling that concepts are published on tuples, the straightforward solution will lead to utilize tuple numbering scheme for the concept naming. In dynamic environment the context can rapidly change and consequently it is necessary to modify the values contained in the tuples. This is accomplished by Lime through tuples publication and elimination. However, the concept can remain unchanged and thus a different naming scheme is really mandatory. For example, if the user is moving in the environment, the actual position changes (i.e. the tuple) but the location concept is unchanged.

3.3. Tuple Header

The tuple header structure is reported in Table 2. The first field is the `Source` that contains the concept URN. The second field, `Relationship`, accounts for the XML hierarchy. It contains the URN of the concept associated with the hierarchically upper XML TAG. For example, if a property is referred to a specific instance of a class then its `Relationship` field will contain the URN of the tuple associated with that class. For the case in which the XML TAG represented by the tuple does not have an upper parent, the 'thing' OWL class with URN=0 are considered as root parent.

Finally, the `Time` field represent the absolute generation time of the tuple that can be used as an index of context data "freshness".

Table 1 – Tuple Header

Field	Value
Source	URN
Relationship	URN
Time	TimeStamp

3.4. Tuple Context Data

The Context Data part can be of three types according to the referenced OWL concept: Class, Object Properties and DataType Properties.

Table 2 describes the context data for the Class OWL concept. It contains just the 'Class Type' field reporting the rdf:ID of the class, i.e. the type of class that has been instantiated.

Table 2 - Class Context Data

Field	Value
Class Type	rdf:ID

Table 3 - DTP Context Data

Field	Value
DTP Type	rdf:ID
Value	rdfs:range

Table 4 - OP Context Data

Field	Value
OP Type	rdf:ID
Value	Reference URN

As example, referring to Figure 3, the tuple associated with the instance of the Person class is:

<Person, URN_P, 0, T_P>

The first tuple field contains the value Person that represents the 'Class Type' field. The "red" fields constitute the tuple header. URN_P is the Source, i.e. the concept URN. The next field, equals to 0, is the value of Relationship meaning that the XML TAG does not get an upper parent. Finally, 'T_P' is used to indicate the Time.

The DataType Properties tuple format is reported in Table 4. The 'DTP type' field contains the type of the DTP; i.e., its rdf:ID. The Value field contains the specific value of the DTP that must be in the rdfs:range defined by the ontology. As example, referring to Figure 3, the tuple for the DTP FirstName is:

<FirstName, Andrea, URN_{FN}, URN_P, T_{FN}>

where the first two fields represent the DTP part and the other fields represent the tuple header. Regarding the header, URN_{FN} represents the concept URN and let us notice that the Relationship field contains URN_P, indicating that the XML TAG mapped by the current tuple is hierarchically below the Person class identified by URN_P.

Finally, Table 5 reports the OP tuple format. The 'OP type' field contains the type of the OP; hence, its rdf:ID. The Value field contains the URN of the class instance to which the OP is referring (i.e., it is a pointer to a concept).

As example, referring to Figure 3, the tuple associated with Location OP is:

<Location, URN_L, URN_{LOP}, URN_P, T_{LOP}>

where the first two fields are the OP part and the latter fields are the header (URN_{LOP} is the concept URN).

3.5. Tuple positioning

Lime allows moving the tuple from an agent to another (e.g., from the Location Provider device to User Device). This facility can be used to reduce the network traffic exploited during the context data search. As a matter of fact, taking as symbolic example the case in Figure 3, if all the tuples related to a person are moved on its user device (e.g. the Location OP and the entire Location class), an application knows a-priori that all the tuples associated with that person will be found on its device; hence, the application can perform unicast Lime queries, instead of broadcast ones, needed when the application does not know where the tuples are.

Anyway, this approach gets a drawback: if we move an entire class containing highly dynamic data (e.g., the DTPs of the Location class continuously changes it value) from the source agent to another one, the former has to frequently update the moved tuples, so increasing the network load. Summing up the previous reasoning, we chose to move only the OP tuples.

3.6. Context Data Retrieval

This section deals about the operations needed to find a context data. The general approach is a sequence of broadcast or unicast Lime queries (driven by the ontology schema), which, step-by-step, conduct us toward the target data.

As example, we consider the case of an application that has to monitor the location of the person named "Andrea". First of all, the application broadcasts a query for the DTP tuple containing the person name. The performed Lime query is:

<FirstName, Andrea, *, *, *>

where '*' means 'any value'. The Lime agent on the user device will answer with the following tuple:

<FirstName, Andrea, URN_{FN}, URN_P, T_{FN}>.

At this point, through the URN_P, the application knows the network IP address of the user device holding the Person class. Thus the application can perform a unicast query toward the user device requiring the Location OP tuple. The query is:

<Location, *, *, URN_P, *>

indicating the request of the Location OP hierarchically below the Person class identified by URN_P. Lime agent on the user device will answer with

<Location, URN_L, URN_{LOP}, URN_P, T_{LOP}>.

Now, the application knows URN_L that allows it to perform unicast queries to the Location Provider agent requesting the Latitude, Longitude and Altitude DTP tuples.

As proof of the effectiveness of the tuple positioning and naming strategies presented in the previous sections, during the data searching, we performed only the first query in broadcast due to a complete lack of knowledge. After the first query we can exclusively use unicast one.

4. RELATED WORK

Various software architectures have been proposed for the context sharing in pervasive environment. The different solutions have to be compared mainly on the communication infrastructure characteristics and on the context data description formalism. In the follow we consider the only solutions that use Semantic Web technologies for context data formalization.

Early ontology based context management approaches relay on a central node; i.e. a server component holds all the context information and reasons on them, like in COBRA [6] or in GAIA [1]. A different approach has been pursued in the "Semantic Space" system, [7], to account the mobility issue. They propose an architecture that uses UPnP general events notification architecture to disseminate the row context data that are collected locally and represented by OWL. These approaches are unfeasible if peer-to-peer agent interactions take place. Moreover, each application has to relay on the central node for reasoning tasks possibly causing a scalability problem.

An interesting work on tuple spaces but not suitable for spontaneous networks, is reported in [8]. The "sTuples" are an extended version of Java Spaces in which semantic tuple matching is introduced by tuples that contain semantically described query.

Other significant works on context management in infrastrucured networks and not directly related to pervasive computing are presented in the follow. A context management architecture targeted for extending the Semantic Web is presented in [9]; "Semantic Web Spaces" system uses a persistent tuple space to share the OWL concepts that are represented by RDF triple. Each agent can collect the context data reading in the tuple space and coping them locally. The architecture is based on XMLSpaces [10] middleware that supports distributed XML statements publication in a tuple space using tuple nesting technique. In the implementation of the Triple Computing vision, a similar approach is used, but the context is exchanged with a new costumed protocol built on top of HTTP [11]).

5. CONCLUSION

The paper presented an architecture for context sharing based on a peer-to-peer service model. The architecture relies on the LIME middleware for data sharing and uses OWL for context formalization. A LIME/OWL Interface allows the ontology concepts to be dynamically shared by the agents that publish, in a distributed fashion, parts of the XML statement representing the context. The XML TAGs representing the OWL concepts are mapped on tuples, shared transparently among agents via the LIME/OWL Interface using the LIME middeware.

The proposed architecture should result as a key for fast system deployment, avoiding the need of a provisional infrastructure. In fact, each software component is directly responsible for the management of its context data without the intermediation of a central server. Moreover, the usage of OWL for context formalization can provide a straightforward integration with Semantic Web technologies

REFERENCES

[1] S. Helal "Programming Pervasive Spaces", IEEE Pervasive Computing, Vol. 4, Issue 1, Jan.-March 2005

[2] A. Ranganathan et al., "Ontologies in a pervasive computing environment", Workshop on Ontologies in Distributed Systems, Acapulco, Mexico, 2003.

[3] G.P. Picco, A.L. Murphy, and G. C. Roman, "Lime: Linda Meets Mobility", Proceedings of ICSE, May 1999.

[4] M. K. Smith, C. Welty, and D. L. McGuinness, Editors, "Web Ontology Language Guide", W3C Rec., 10 Feb. 2004.

[5] Z. Li and M. Parashar, "Comet.. A Scalable Coordination Space for Decentralized Distributed Environments", Proceedings of HOTP2P'05, 21 July 2005, La Jolla, CA, USA.

[6] H. Chen et al. "Semantic Web in in the Context Broker Architecture", In Proceedings of PerCom 2004, March 2004.

[7] X. Wang et al. "Semantic Space: an infrastructure for smart spaces", Pervasive Computing, IEEE, Vol 3, Issue 3, July-Sept. 2004, pp. 32 - 39

[8] D. Khushraj et al. "sTuples: semantic tuple spaces", Mobile and Ubiquitous Systems: Networking and Services, 2004, Aug. 2004, pp. 268 - 277

[9] R. Tolksdorf et al, Semantic Web Spaces "Technical Report TR-B-04-11", Freie Universität Berlin, Germany, July 2004

[10] R. Tolksdorf et al "XMLSpaces.NET: An Extensible Tuplespace as XML Middleware" Technical Report B 03-08, Freie Universität Berlin, Germany, 2003.

[11] D. Fensel, "Triple-based Computing", Technical Report DERI-TR-2004-05-31, Digital Enterprise Research Institute, Ireland, May 2004

ACKNOWLEDGE

This work is part of the Virtual Immersive COMmunication (VICOM) project founded by the Italian Ministry of University and Scientific Research (MIUR-FIRB projects).