Mapping for Activity Recognition in the Context-aware Systems using Software Sensors

Kamran Taj Pathan, Stephan Reiff-Marganiec, and Yi Hong
Department of Computer Science
University of Leicester
Leicester, United Kingdom
ktp2, srm13, yh37{@le.ac.uk}

***Abstract*—Context-aware systems are concerned with identifying the context of a user and then to either provide that information based on queries or to automatically decide on appropriate actions to be taken. Some context aspects (such as location) are easy to sense through hardware, while the activity of a user has shown to be somewhat elusive to being sensed with hardware sensors. As users use web services more frequently they are exchanging messages with the services through the SOAP protocol. SOAP messages contain data, which is valuable if gathered and interpreted right – especially as this data can be shedding information on the activity of a user that goes beyond "sitting at the computer and typing". We have developed software sensors, essentially based on monitoring SOAP messages and inserting data for further reasoning and querying into a semantic context model. In this paper we consider a solution to map the data from a SOAP message to our OWL ontology model automatically. Specifically, we explain the methodology to map from SOAP messages to an existing structure of knowledge.**

Keywords: context-aware systems, software sensors, activity recognition, mapping from SOAP to OWL

I. INTRODUCTION

Context-aware systems have been researched and are also being used for a while – often to provide location aware services to users (such as targeted advertising for mobile users through SMS messages offering discounts at restaurants in the vicinity of the user). However, sensing the activity of the user (i.e. what a user is doing?) has been very hard to tackle. Most of the context gathering is achieved through the use of hardware sensors (e.g. tracking devices etc.). The systems based on software sensors only process the user data according to web sites visited or relevant documents [2][6] or tell the online status [12] which is either updated by the user or automatically through logging in or out. The activities a user undertakes otherwise are not understood. Some of the advertising services (such as Google in its Gmail interface [9]) look at the description and subject of the email and advertise; no matter if that advertisement is necessary for the users at their situation or not and hence might be to no avail.

A typical understanding of a sensor is "a device that converts a physical phenomenon into an electrical signal. As such, sensors represent part of the interface between the physical world and the world of electrical devices, such as computers" [13]. This view has a very hardware oriented angle. Hardware sensors typically measure a specific phenomenon such as a GPS location or the ambient temperature. However, they could be more complex, for example a camera might track a user's position and through a connection to models can identify whether a user is sitting or standing or even where on a screen he is looking. It should be said that for the latter to work relatively complex setups are needed and hence a high cost concurs with the deployment of hardware sensors. In addition, it is not really possible to conclude what exactly a user is doing: they might look at their email client, but are they reading private or work emails, and which work project does the email relate to? Understanding the latter automatically could lead to very fine grained understanding of the time projects take.

Returning to the view of sensors as devices that capture some physical phenomenon, one could argue that it is a very narrow view. And indeed, a sensor could also be software, as long as it acquires data that captures real world snapshots [10].

In today's world there is much exchange of messages between user's and the networked applications that they use; in the context of service-oriented computing this exchange often involves structured messages defined in the SOAP protocol, which contain data on the service invoked, the operation of that services requested and possible a payload of attributes send to the service or received in reply. Using software sensors, we can capture this wealth of data and put it to use in understanding a user's activities as well as in supplementing data gathered from hardware sensors. The challenge is to ensure that the raw data will be converted into structured data (typically populating a context model) where it then becomes knowledge, which can further be reasoned about using some inference mechanism.

For example, if a user is using a Calendar Service and she is sending data for adding an event the data send will make explicit that it is an *AddEvent* request to a *calendar*, and possibly one can even see details

such as the title of the event, the time (from-to), the location, etc. This data is carried through a SOAP message and is usually interpreted by the receiving service. Our approach is to extract that data and store it in an RDF triple form so that automatic reasoning can be performed. For example from event data obtained from a calendar, weather/ traffic conditions of the location and the location of a speaker one could deduce whether the event is likely to start on time.

For this to work effectively we need to instantiate and update an ontology with new data. This paper introduces a novel mapping methodology that can map data from the XML payload in SOAP messages to an existing OWL Ontology and update the instances along with the hierarchy it is based on should there be a need. To place this methodology in context, the respective context model detailing the data model to store and process the data and the infrastructure for context-aware systems using software sensors to sense the activity context is also presented.

This paper is structured as follows: section II presents an overview of the architecture and section III details the context model. Section IV is concerned with the main contribution of the mapping methodology and section V presents an example. Related work is discussed in section VI, and the paper wrapped up with a summary and discussion in section VII.

II. Activity Context Architecture for Context-aware Systems Using Software Sensors

While users are interacting with the services they use, messages are exchanged between the user and the service which contain data. In the context of web services, this data is either in the form of SOAP messages or of a RESTful nature – both are essentially plain text messages on the back of HTTP. To make use of the transferred data and transform it into a useful form we present the service-based activity context architecture which is targeted specifically at software sensors. Figure 1 presents the architecture, which combines traditional web services with semantic web technologies to allow for reasoning on context using both software sensors, but also hardware sensors if available (the latter would simply add additional data to the context model).

In the architecture the user's context information is extracted from the exchanges occurring between user and services, whenever any user is using any kind of service. As part of their natural interaction they are providing data and this data can be harnessed. We achieve this through software sensors capturing the exchanges and extracting data through the context acquisition module. This raw data is further processed in the mapping module to be inserted into the ontology model – the mapping is the key focus of this paper and will be explained in the

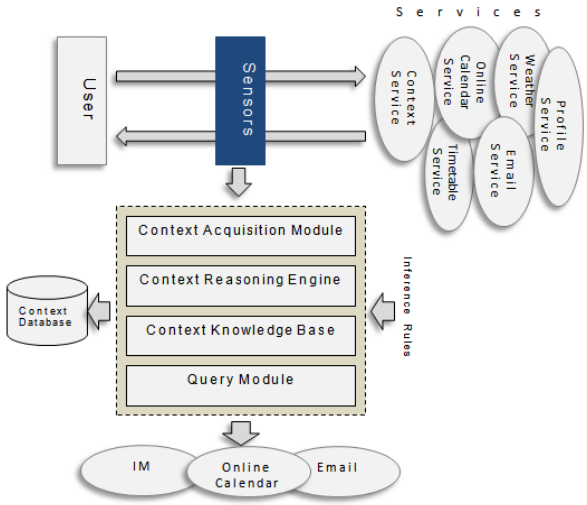

Figure 1. Activity context architecture for context-aware systems using software sensors

next sections. Once data exists in the context model semantic web technologies can be used to query and reason about the activity of the user. This is possible, as the raw data will have been given structure and thus really provides knowledge.

Successful activity sensing using software sensors requires 4 problems to be addressed. These problems are a) placing software sensors which monitor SOAP messages and extract the data, b) mapping that data into a context model form, c) inferring knowledge out of the gathered data, and d) querying the knowledge base to answer queries about user's context.

III. Context Model

The concept of semantic web provides standards for structured knowledge management. In our setting we gather data from services, but need to manage that data and enrich it by reasoning about it to gain information which in turn allows to answer user queries. We have chosen to model our context using OWL [3], as OWL is a W3C recommendation for structural knowledge and hence is widely supported. Specifically, for us it provides the interoperability between context-aware systems and services by defining entities and properties and relations between them. It also provides additional vocabulary by using inference mechanisms based on available reasoning tools.

Figure 2 provides an overview of our context model, which is designed for context-aware systems environments where there is a specific need to sense the activity of users. For example `Place` is the super class of `Location`, because we believe that further descriptions are needed to characterize a location, such as the purpose it serves. Consider a room where

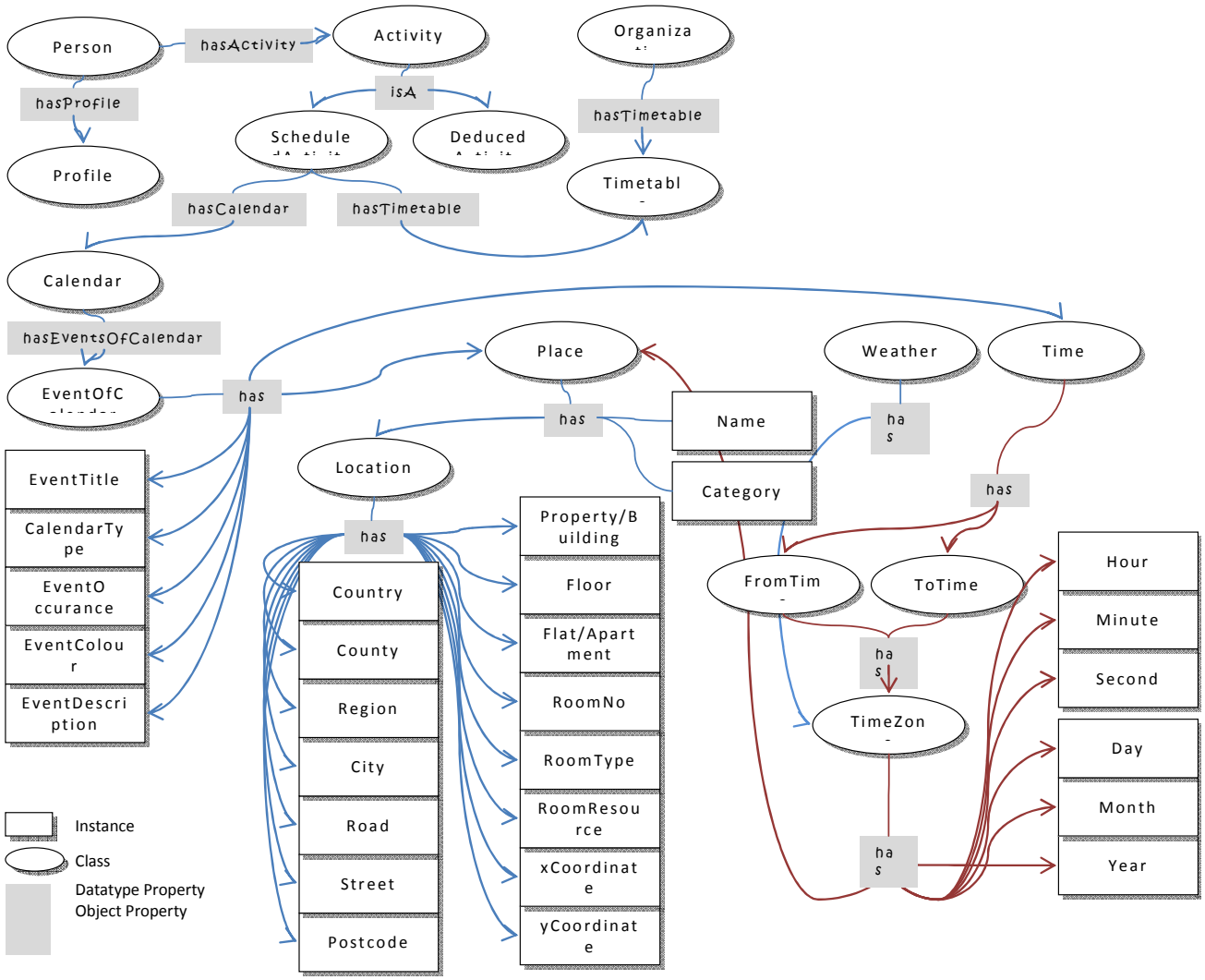

Figure 2 Context ontology for context-aware systems using software sensors

a user might be: the resources of the room can also affect the activity of the user (and if these resources are connected to services they can provide more data!). A teacher in a class room having a white board, multimedia projector, 100 seats capacity and with 100 computers might be offering a lab or lecture – and which can be quite crucial when considering whether interrupting her is possible.

Software sensors pose specific challenges for the context model. In general hardware sensors are much better understood and the specific they measure are more precisely characterized (locations, temperature, light, position of a user) – software sensors on the other hand measure non-physical properties which might be very specific to an activity and hence are more difficult to characterize.

IV. MAPPING

This section considers the core contribution of the paper: the mapping of XML data extracted from SOAP messages to data in the context model.

The input to the mapping task is raw XML data and the output would be a context model populated with the new data gathered. XML files contain tags and values and their relation expressed through nesting and the general tree structure, while in the OWL context model we have classes, instance data and explicit relations expressed through properties. There are obvious mappings from tags and values to classes and instances at a conceptual level (see Related Work for details), however at the concrete data level considered here we require novel techniques.

To automate this matching, we use a syntactical comparison, enriched with semantic information. We

can distinguish four cases, based on how well the tags and ontology instances align. The simplest scenario is where the XML tag name is identical to the instance name in the ontology (note that we assume that syntactically identical terms are referring to identical concepts in this domain; however the context (such as super or sub tags) in which the tag occurs in the XML file could provide further insight). This case will be referred to as a *direct match* and we will formalize this later. The more interesting cases are those where no direct match is found between the tag and the ontology. One way to address this could be to create new instances and extend the context model; however having a stable structure is more suitable for reasoning so we are looking at methods of identifying existing instances with which the new data can be directly associated. We identify two possibilities of creating this association: a synonym of the tag matches with the ontology concept (a *synonym match*) or the hierarchical context of the word contains the ontology concept (a *hyponym match*). The fourth case is covering all those concepts that occur in terms and cannot meaningfully be matched to the ontology.

A. Methodology

Our methodology considers the cases in turn and inserts the value from the XML document into the most appropriate matched ontology instance as shown in Figure 3. Briefly, the approach goes through 3 steps:

1. Attempt to match the tag with the instance; if found then insert the value as a literal and finish. If not found go to next step.
2. Identify synonyms for the tag and match each synonym with the instance; if found then insert the value as a literal and finish. If not found go to next step.
3. Find the tag in a lexical database (such as WordNet) providing a hierarchy (that is hyponyms). Match each hyponym with the instance; if found then insert the value as a literal and finish. If not found go to next step.

Note that for step 3 we could consider an alternative of enhancing the ontology with elements along a path in the hierarchy. While this would require extra effort in reasoning, we at least know that the expansion is conducted in a structured way and the structure of the lexical database could be used as an extra factor in reasoning. We will now define the four levels of match in detail.

1) Direct Match

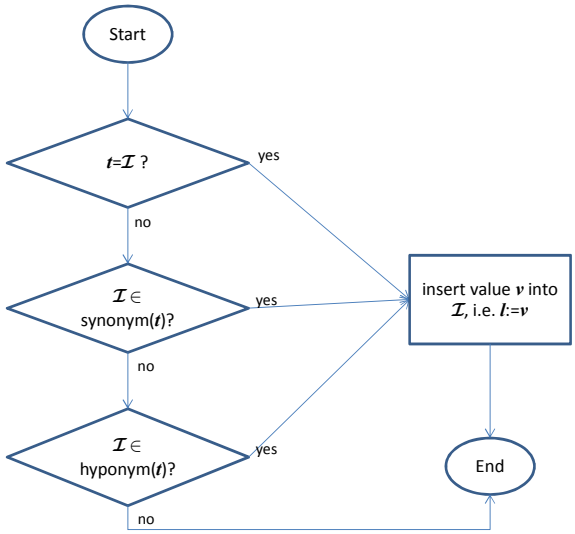

Figure 3. Mapping Methodology

The easiest kind of match is one where the tag matches the instance of the ontology (see also Figure 4), and in this case we wish to insert the value from the tag directly as literal associated to the instance:

Definition: Direct Match
Let

- $\mathcal{I}$ be an instance of an ontology $\mathcal{O}$,
- l be a literal of instance $\mathcal{I}$,
- t be a tag in an XML schema $\mathcal{X}$, and
- v be a value for t

then

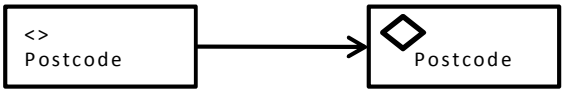

Figure 4. Direct Match

$l = v$ **iff** $\mathcal{I} = t$.

2) Synonym Match

The second type of match is where we find that a synonym of the tag (for example location and position can be seen as synonymous in this context, see Figure 5) matches the instance in the ontology.

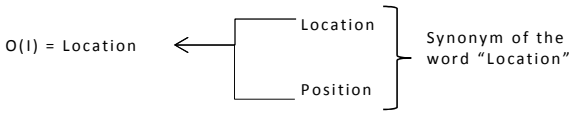

Figure 5. Synonym Match

Definition: Synonym Match
Let

- $\mathcal{I}$ be an instance of an ontology $\mathcal{O}$,
- l be a literal of instance $\mathcal{I}$,
- t be a tag in an XML schema $\mathcal{X}$,
- v be a value for t
- *synonym(t)* be a set of synonyms for t as obtained from a lexical database.

then

$l = v$ **iff** $\mathcal{I} \in$ *synonym(t).*

3) Hyponym Match

The third type of match is where no direct match and no synonym match can be found. Existing work either merges tags by looking at the super-tags and inserting that value (thus assuming a semantic relation between the super and sub-tag. For example, if a person tag has a position sub-tag and position would not exist, the value would be inserted with a person class, rather than with location as would be desirable. The alternative is to create a new ontology model.

Both are not suitable here – the former for reasons indicated above and the latter because we have a fixed context model to fulfill the needs of a context-aware systems making use of software sensors and hence we need to instantiate that model.

The proposed hyponym match is generic and could be used for any purpose where XML data needs to be inserted into fixed ontologies.

Definition: Hyponym Match
Let

- $\mathcal{I}$ be an instance of an ontology $\mathcal{O}$,
- l be a literal of instance $\mathcal{I}$,
- t be a tag in an XML schema $\mathcal{X}$,
- v be a value for t
- *hyponym(t)* be a set of hyponyms for t (that is the branch of the tree in which t is found) as obtained from a lexical database.

then

$l = v$ **iff** $\mathcal{I} \in$ *hyponym(t).*

The hyponyms are obtained from a lexical database (e.g. WordNet) and checked for match with the instance; if a match is found the value is inserted. The hyponyms are processed in order, going upwards in the hierarchy of words and the first match will be chosen (as clearly that will be closest in meaning to the tag).

4) No Match

We can arrive at a situation where none of the above steps can achieve a match for the data item from the XML file. While it seems desirable to match as much information as possible, but there might be data which simply does not add to the context and hence it is safe to ignore that. Furthermore, even if a data item could be adding to the context, one should keep in mind that any context data gathered is often not 100% accurate and could be contradictory to information simultaneously gathered, changes quickly and hence has a limited period of validity and is possibly made more or less reliable through reasoning anyhow. In the light of these factors missing some items can be seen as a negligible problem.

V. SAMPLE SCENARIO

A. The Supervisor and the Research Student

A research student wants to meet with his supervisor to discuss his work without prior arrangement, but does not know about the availability of the supervisor. He is of course aware that the supervisor is busy, with involvement in research projects and teaching as well as administrative tasks.

Meanwhile, the supervisor is working on computer on one of his research projects and does not want to be interrupted; however he has not updated his current context.

In these kinds of real world situations one is not only concerned if the person is available – actually available can have many meanings (someone is "in", "free", "could be disrupted in an emergency", etc.). One could update ones current context which is tedious and needs to be repeated every time the context is changed (and hence probably is not done at all), or provide some means through which the context is automatically updated and can be queried by authorized users – this work clearly supports the latter.

B. Queries

We have discussed the context model and the insertion of context earlier, so here we are looking at making use of it. A typical question to ask in the context of the scenario might be "**What is my supervisor doing on 22nd Feb 2011 from 09:00 to 11:00?**"

We assume here that the person asking, that is the research student is authorized to ask about the teacher.

Formally, as we are using an RDF based ontology we can ask questions such as this through the SPARQL query language:

```
SELECT ?Faculty ?Activity
WHERE ?Time
```

This SPARQL query looks very much like an SQL query, and asks about a faculty member's activity at a particular time – some of the arguments (values after the ?) could be instantiated with specific values, directly filtering the results to those matching.

C. Rules

Simple queries asking about the data stored in the model are easy to formulate and ask, but do not allow to harness the full power of the semantic technologies at hand. We can enhance the richness of questions to be asked by allowing the use of additional reasoning rules that can combine existing facts to derive new facts. Many of these rules can be quite specific to certain domains, reflecting interpretation of data in those domains. For example, in University there might be sufficient information on rooms and schedules available to derive that a Professor being in a teaching room in the postgraduate block is probably teaching a Postgrad Class. Here are some of the rules that we used in the context of the scenario:

1. $Place(f,p) \wedge Time(t) \Rightarrow teachinClass(f,c)$

   where *f* is a faculty member, *p* is place name or name of a building, *t* is a time zone and *c* is a postgraduate class.

2. $Timetable(f,o) \wedge Calendar(f,pc) \Rightarrow teachinClass(f,c)$

   where *f* is a faculty member, *o* is the organizational calendar (including a timetable), *pc* is the personal calendar and *c* is a postgraduate class.

3. $Timetable(f,o) \wedge Weather(l) \Rightarrow teachinClass(af,c)$

   where *f* is a faculty member, *o* is the organizational calendar (timetable), *l* is a location, *af* is another faculty member (distinct from *f*) and *c* is a postgraduate class.

Considering Rule 3, we say that if a staff member is foreseen to teach a certain class by the organisations timetable, but the weather conditions are adverse, an alternative faculty member might conduct that class.

D. Degree of Confidence

Having gathered data from different sources and being able to derive new conclusions on them we should turn our focus to the issue of how certain we can be about the information derived. We term this concept *degree of confidence* as it should show how certain an enquiring user can be that the information she receives truly reflects the activity of the entity enquired about. This could for example be quite crucial if billing of consultants time or travel arrangements to meet someone will be based on such activity sensing.

Returning to our example, a faculty member might have private and work calendars. If they are inserting events in their work calendar which reflect working time activities, these are probably accurate for day time. However, social activities would usually take place outside working hours and hence the private calendar might be more accurate for such activities. In large organizations there might be further departmental and institutional calendars to be considered. So to identify what a user is doing, it might be worthwhile considering which data source is most appropriate and should take priority in case of events being in each of them.

At this stage we propose a simple ranking of data sources, which is dynamic over time in that different preferences are given at different times.

Following our example, assume the user to have an Organization Calendar (*OC*), a Work Calendar (*WC*) and *a* Personal Calendar (*PC*) we get the following ranking:

```
If (time is 8:00am and 5:30pm) and
(day is Mon to Fri) then
            OC > WC > PC;
Else
            PC > WC > OC
```

This states that during the normal working hours the organization and work calendars are more reliable than the private calendar, while out of working hours the calendars that are under the user's control gain in relevance.

One could consider a more fine grained approach, and indeed this is one of our aspects of further work. For example we could strengthen the confidence on the activities is during mapping from XML to OWL, based on specific instances and also on how certain we are about the match that has been made when inserting data.

VI. Related Work

There is some existing work addressing the problem of mapping from XML schemas to OWL ontologies (or their respective RDF descriptions). This work falls into two broad groups: work that creates ontology models based on the XML schema [1][7] and work that matches an XML instance document into a related pre-existing ontology [11]. This body of work provides a sound understanding of the relation between elements in XML and their ontological counterparts, and we summarize the key relations in Table 1 and Figure 6.

TABLE 1. RELATED WORK FOR MAPPING FROM XML TO OWL

XML	OWL
Represented by Trees with Labelled Nodes	Represented by triples (i.e. Subject-Predicate-Object)
Tree Structure	Concept hierarchy
Schema	Model
Instances	Instances
<tag>	Resource/Literal (depends if the tag has a value)
Nested Tags	Resource with Object Properties (if the tag has no value)

This existing work maintains the class hierarchy given through the tree structure of the XML documents without maintaining their actual structure. In relation to our work, these issues become somewhat preliminary as they might help to establish the initial ontology models. The second benefit is the conceptual relation of the concepts. However, for our work we wish to reuse existing context models and are more concerned with mapping the actual XML instance data into the context model. For this we need, and this is the contribution of this work, a way to identify the right model element to assign data to and map the actual data. We also do not wish to extend the ontology model with new concepts, but rather keep that stable, while at the same time being able to insert as much information on XML schemas that are encountered as part of services exchanges. Thus the problem addressed here is more concerned with matchmaking at a vocabulary rather than structural level and hence the need to use a lexical database, such as WordNet [4], arises. so that we can check and acquire the results accordingly.

VII. CONCLUSION AND FUTURE WORK

We have identified that hardware sensors are an expensive way to properly identify user activities, and that their use is not perfect. As much of a user's activity nowadays takes place across networked services, we identified that the exchanged data can enrich the information available about user activities and hence propose software sensors as well as an architecture of how they would be used in a context aware system. In this context, and using our existing context model we have presented a novel solution to the problem of associating data from SOAP messages with elements in the context model. This technique

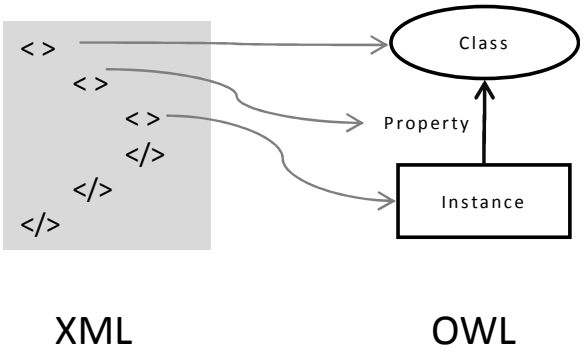

Figure 6. Relationships of XML elements and OWL objects

which maps data from the XML payload in SOAP messages into an existing OWL model which then allows further reasoning rules to use that data to answer user queries.

Current work is concerned with refining the implementation of the test system and conducting a more realistic case study. Future work will consider whether we can also use the data that is observed but is not captured by the current mapping process as well as investigate richer ways to compute confidence in observations.

REFERENCES

[1] Bohring, H., & Auer, S.: Mapping Xml to Owl Ontologies. Leipziger Informatik-Tage, **72** (2005) 147-156

[2] Budzik, J., & Hammond, K. J.: User Interactions with Everyday Applications as Context for just-in-time Information Access. (2000) 44-51

[3] Dean, M., & Schreiber, G. (eds): OWL Web Ontology Language Reference. W3C (2004). Available at http://www.w3.org/TR/2004/REC-owl-ref-20040210/, last accessed 14 Oct 2011.

[4] Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press (1998).

[5] Ferdinand, M., Zirpins, C., Trastour, D.: Lifting XML Schema to OWL. Web Engineering, (2004) 776-777.

[6] Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G, & Ruppin, E.: Placing Search in Context: The Concept Revisited. (2001) 406-414.

[7] Ghawi, R., & Cullot, N.: Building Ontologies from XML Data Sources. (2009) 480-484

[8] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H.F., Karmarkar, A., & Lafon, Y: SOAP Version 1.2 Part 1: Messaging Framework (2nd Edition). W3C (2004). Available at http://www.w3.org/TR/soap12-part1/, last accessed 14th October 2011.

[9] Gmail. https://mail.google.com/.

[10] Loke, S.: Context-aware pervasive systems: Architectures for a new breed of applications. Auerbach Pub (2006).

[11] Rodrigues, T., Rosa, P., Cardoso, J.: Mapping XML to Existing OWL Ontologies. (2006) 72-77.

[12] Truong, H. L., Dorn, C., Casella, G., Polleres, A., Reiff-Marganiec, S., & Dustdar, S.: inContext: On Coupling and Sharing Context for Collaborative Teams. (2008) 225–232.

[13] Wilson, J.S.: Sensor technology handbook. Newnes (2005).