User Activity Recognition through Software Sensors

Stephan Reiff-Marganiec, Kamran Taj Pathan, Yi Hong

Department of Computer Science
University of Leicester
Leicester, United Kingdom
{srm13, ktp2, yh37}@le.ac.uk

Abstract

Context-aware systems are an instance of the ubiquitous or pervasive computing vision. They sense the users’ physical and virtual surrounding to identify the best system support for that user and adapt the system behaviour accordingly. The overall architecture of a context aware system can be broken into a number of logical aspects: gathering context data, storing the data, deriving knowledge through reasoning and mining and retrieving that knowledge to finally adapting system behaviours. Context is anything characterizing the environment of the user – their location, the ambient temperature, the people they are with, their current activity and some even consider the user’s mood. Traditionally context information has been gathered through the use of hardware sensors, such as GPS sensors or smart badges for locations and there has been work to track user’s eye movements at their desk to see which application they are using. However, determining the activity of a user has shown to be 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 propose a complimentary sensor technology through the use of software sensors. The software sensors are essentially based on monitoring SOAP messages and inserting data for further reasoning and querying into a semantic context model. In this paper we consider details of extracting the data from SOAP messages in a non-obstructive way and show a solution to map the data from a SOAP message to our OWL ontology model automatically. On the latter, we specifically explain the methodology to map from SOAP messages to an existing structure of knowledge.

Keywords: context-aware systems, software sensors, activity recognition, data mapping

Introduction

Context-aware systems have been on the research agenda for quite a while, and there are many aspects of such systems readily used today. Typical systems often provide location aware services to users (such as targeted advertising for mobile users through SMS (Short Message Service) 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 [1][2][3] or tell the online status [4] 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 [5]) look at the description and subject of the email and advertise; no matter whether that advertisement is relevant for the users in their given situation and hence possibly missing the intention of advertising.

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” [6]. This view has a very hardware oriented angle. Hardware sensors typically measure a specific phenomenon such as a GPS (Global Positioning System) 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 complex setups are needed and hence a high cost occurs 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? This is especially relevant in a world with knowledge workers working on many projects simultaneously, answering emails and phone calls relating to each of these rather than just focusing on one task. Understanding the time spent on specific tasks automatically could lead to very fine grained understanding of the time projects take and hence will allow for very precise invoicing to clients and good predictions for efforts for new projects.

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 [7].

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 (Simple Object Access 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, invited participants, 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 (Resource Description Framework) 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 a) be able to capture the SOAP message and extract data from it and b) instantiate and update an ontology with new data. This chapter addresses both aspects. We discuss the data extraction and introduce a mapping methodology that can map data from the XML payload in SOAP messages to an existing OWL (Web Ontology Language) Ontology and update the instances along
with the hierarchy it is based on should there be a need. The latter is building on work presented in [8]. 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. Figure 1 shows the overall architecture employed in our work, with this chapter focusing on the sensors and context acquisition aspects shown in the figure.

![Figure 1: Architecture for Context-aware System using Software Sensors](image)

This chapter is structured as follows: the Background section will discuss developments of sensors, context models and XML (Extensible Markup Language) to OWL mappings. We will then present our adopted (and adapted) context model and discuss design considerations and knowledge generation by reasoning in relation to the model. The following two sections introduce the data extraction and the data mapping methodology. We conclude with a summary and an outlook on future directions and open questions.

**Background**

There has been a growing interest in context-aware systems, significantly fostered by the way in which people use computing devices these days. In an always connected, always on environment with smart phones and other mobile devices making their way into our lives in ways overshadowing the impact of desktop and home computing there is a desire by manufacturers to provide differentiating features and by users to seek ever new applications and services. In addition to context-aware, a number of other terms have been used and these are often overlapping when it comes to end-user systems. Ubiquitous computing [9] or pervasive computing [10] refer to the collective use of computers available in the physical environment of users, perhaps embedded in a form invisible to users representing a vision for putting computers out into everyday life and environment, instead of representing the environment in the computers. Looking at embedded computers in cars and internet enabled TVs and refrigerators this is already much of our daily reality now seeking new ways of making the user’s life easier. Sentient computing [11] refers to systems “using sensors and resource status data to maintain a model of the world which is shared between users and applications”. As such systems try to build a model of a part of the
world from sensory information about the users’ circumstances and environment. This idea is very closely related to context-aware computing, but with an emphasis on the world model [7].

Existing context-aware middleware and applications provide and exploit various types of contextual information about location, time, user activities, user’s preferences, profiles of users, devices and networks, etc., [12][13][14]. Projects such as ECOSPACE [15] or inContext [16] provide middleware platforms taking the context-aware applications into collaborative work environments.

What is common in all of these applications is a need to gather context data as the most basic information to drive the whole system structures. Of course it is not just gathering this data, but also storing it in ways that allow knowledge generation. We will now look at the aspects that are most pertinent to the work presented: sensors, context models, and data to knowledge mapping (for XML and OWL).

**Sensors**

According to Klein, a sensor is a transducer (front-end hardware) that “converts the energy entering its aperture into lower frequencies from which target and background discrimination information is extracted in the data processor” [17]. A likewise very physical definition is provided by “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” [6]. Interestingly the latter continues to take this full circle, by also considering the way the sensing system can influence the environment: “The other part of this interface is represented by actuators, which convert electrical signals into physical phenomena” [6].

However, more generically a sensor is a device (in the widest sense) that can acquire data about the environment. In that wider sense, sensors might be hardware or software, or indeed the combination of both. So, for example a location tracking device or the computer clock accessed using an operating system function would both be considered sensors (one for location, the other for time) [7].

Using sensors to gather context information, also called context sensing, refers to the process of acquiring context information. Fersha et al. [18] define two types of context sensing mechanisms:

- **Event-based sensing** monitors the environment for occurrence of events and sends notification to the applications upon detection of an event.
- **Continuous sensing** explores the real world information continuously and provides streams of observations.

Both of these two types have advantages and disadvantages and one might be more suitable than the other to specific scenarios. Considering sensing the context of a taxi as an example, one could say that the location of the taxi is best sensed in a continuous fashion (as it constantly changes if the taxi is moving) while the fact whether the taxi is occupied or not is more event based (a passenger getting in or out). There are a number of interesting questions regarding handling the data amounts occurring in such settings. For complex event processing [19] is being employed to deal with streaming data, and the work in [20] focuses on determining ways to provide sensor data to applications with essentially zero latency.

In general when considering continuous sensing there are questions as to how often sensors shall be polled or at what frequency data should be sent – sending all information to all users is not possible due to
the vast amount of data generated. What makes this more complex is that context is more or less dynamic, consider for example locations: these are very dynamic for a person but quite stable for a desktop device or printer, and absolutely fixed for rooms. The current approach is to preset different polling rates by the user or by the application at the start up [21]. These polling rates are based on people’s experiences. Brewington and Cybenko [22] develop a formal model for monitoring information sources by using previous observed information lifetimes to estimate change rates. Their model was developed for Web pages but is applicable to many information-monitoring problems, including context sensing using software. More recent complimentary work formalised a push-pull architecture with a filtering mechanism that specifies when services or sensors should deliver certain types of information [23]. Context-sensing techniques need to be reliable and robust, which is sometimes very hard to achieve: consider active badges [24] as an example. If the badge is not worn by the user but left in the office then the location determined by the system will be false. This requires systems to be build in such a way that information has lifetimes (old context data is probably less reliable), confidence of correctness (some data is more trusted) and redundant (several sensors should provide insight that jointly allows to arrive at conclusions).

A study by Clarkson [25] looked at gathering data using sensors in real life with a view of identifying patterns. Data was collected using two cameras, a microphone, and an orientation sensor worn by a user for 100 days – all of these are of course again hardware sensors. Quantitative analysis techniques were used to extract patterns in daily life showing that a “person’s life is not an ever-expanding list of unique situations”[25], but that there is a regularity in daily life’s situations. Such regularity is of course what allows for data mining and reasoning techniques to generate knowledge from context data successfully.

As we mentioned earlier, systems based on software sensors are very rare – and often they are about adapting web sites or using online status information, rather than the much wider use of and more rigid definition of sensors proposed here. Existing work uses user data based on web sites visited or relevant documents [1][2][3] which is automatically gathered e.g. through shared cookies. In case of using online status [4], the user will either have to update this manually or it is changed through logging in or out. In the Mobisaic Web Information System [26] context information is extracted from the dynamic URLs, allowing for documenting which web pages change automatically by looking at the change of other variables.

**Context Models**

A well-defined data model helps with aspects of processing and storing data more effectively, and this also holds for context-aware system. The data concerned here is of course context data which has a complicated structure and intricate links that are often exploited not only by the structure of the data model, but also by superimposed reasoning rules which generate the more interesting knowledge. Additionally the system often is distributed, so different parts of the system which might have been written independently will access and store data in the same data store. There are several models in existence, often specialised on the main context aspects that the researchers are concerned with. Furthermore, the approaches use different modelling techniques, leading to distinct advantages making comparison more complex. We will now look at a number of models used to represent, store and exchange context data.
Schilit et al. [27] used key-value pairs to model the context by providing the value of a context entity to an application as an environment variable. Markup scheme models are based on a hierarchical based data structure which extends on this simple model by using tags with attributes and content. They are usually based upon XML type languages such as RDF/S (RDF Schema) and have the advantage of easy tool access, but lack of formality and expressiveness. The Unified Modelling Language (UML) is a language to model the context using UML diagrams. Bauer et al. [28] and Henricksen et al. [29][30] defined a context model for in air traffic management in a graphical way using UML. The resulting object oriented models provide encapsulation and reusability, where object level details are encapsulated (hidden) to other components. A first logic based context modelling approach has been published by McCarthy [31], introducing context entities as abstract mathematical entities which are helpful in artificial intelligence. Ontologies are commonly used to specify concepts and their interrelations [32] and context models based on ontologies have been first proposed by Otzturk and Aamodt [33] and have since proven to be useful and popular in this domain.

[33] proposed an ontological model because of a need to normalize and combine knowledge from various domains and thus formalize it and allow to reason about the knowledge. The context of this context model was not context-aware systems, but rather a psychological study depending on contextual information. The CoBrA system [34] uses a broker-centric agent architecture to provide runtime support for intelligent meeting rooms. It offers a set of concepts based on ontologies to characterize entities such as places, persons and objects. Most of this work focuses on describing people and their locations; it also considers time. However, it does not consider the activity or the motivation (the why).

Gu et al. [35] also followed an ontology-oriented approach by developing a generic context ontology. This model is quite complete in its own way, but this lacks the division in upper and lower ontologies. This decision, trivial as it might sometimes seem, allows to distinguish generic concepts from more domain specific ones and thus eases specialization of ontologies. For example, in contrast to our work, activity context is not considered in much detail in their work. However, activity context is a very rich source of information for advanced adaptation as was shown for collaborative systems in the inContext project [16].

What has crystallized over the last decade is that ontology based approaches for context modelling are very promising as they offer the needed structure, while providing ease of extensibility, reusability. All this is provided at a level of formality open to reasoning while still providing simplicity offering a good understanding of the models. There are a number of standard languages for defining ontologies, predominantly the OWL, which provide ready access to tools and also readily tie in with web services (in fact research on Semantic Web Services uses OWL at its heart to introduce semantics to what are otherwise just syntactically described services).

**XML to OWL Mapping**

One need specific to software sensors is the need to convert from data observed in structured forms to other structured forms. Specifically, we require methods that can map data from XML payloads in SOAP messages to OWL ontologies. There is 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 schemas [36][37] and work that matches an XML instance document into a related pre-existing ontology [38]. 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 2.

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

**Table 1:** Relationships between XML elements and OWL elements

![Figure 2: Mapping XML elements to OWL](image)

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 a contribution presented later, 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 [39], arises, so that we can check and acquire the results accordingly.

**Context Model and Reasoning**

Context modelling specifies the context related entities and relationship, and constraints between each other. Taking an example of our scenario, a user’s activity could be derived from his profile, calendar, timetable and email services. Context model provides the structure and methods to process the context
data, which can be saved for later use. Context is quite wide ranging and includes, for example, a user profile information, her location or planned activities, but generally it is quite varied Error! Reference source not found.. After acquiring the user’s context data needs to be stored (at least while it is valid and of use) and it will be reasoned upon or mined to extract knowledge to provide information needed by the context-aware systems or its users. A number of context-aware computing applications need information to be exchanged between entities, which might be user or services, and the context model should also support that interoperability.

We presented a generic semantic model for use with software sensors in a service-based context-aware environment. It provides general upper level classes, properties and relations between them so that lower ontologies can easily be created for particular domains. Such a generic model of context is beneficial for systems or services which are interacting with each other and the end users, who ultimately try to accomplish their daily tasks more easily in the context-aware environments utilizing software sensors. The use of standard representation languages makes sharing, reuse and extension of context models and context data easy. In addition, the model provides the ability to infer new knowledge from gathered context information.

The generic model of context presents the context in a form which provides general level upper classes, properties and relations such as Place and Resources capturing the aspects which are always useful when considering context information, independent of domain. These can then readily be extended using lower ontologies for specific domains, maintaining the overall structure and relationships (if one wants: conservatively extending the model). The model will allow to retrieve knowledge, which will be at higher levels of abstraction than the data gained from the sensors; when considering activities we like to obtain information on the “situation”.

**Design Considerations**

The Semantic web provides standards for structural, syntactic, meta-data and ontology languages. We have selected the Web Ontology Language (OWL) to develop our context model because it is an international standard for information representation with good tool support. OWL allows interoperability between systems and services, provides inference mechanism by providing additional vocabulary, defines taxonomies, properties, relations and constraints. The fundamental concept for context organisation in context aware systems is W4H [40]: who, where, when and what, which are according to context entity, place, time and activity respectively and how (the devices used). Essentially W4H contains all the questions one might ask about the context. The “Where”, that is location, has seen much attention in terms of its use and acquisition. This has resulted in great successes with most current context aware applications in fact being location aware applications. “When” is, time zones apart, fairly trivial. Both “Who” and “How” is often captured by people and resource profiles – however these two ideas can be more complex. Our model, not surprisingly, includes classes for all of these concepts. However, with devices getting more personal, we decided to consider how and who together as part of our entity class. Thus we identify 4 main high level components of context for our model:

**Entity (who and how):** An entity which performs some action and can be an actor, an organization or an object (machine or another service etc.).

**Place (where):** A place is the physical location in a pervasive environment.
**Time (when):** The time is usually tagged against interactions and observations in the system.

**Activity (what):** Any information that allows to capture the user’s activity occurs in this category.

We are most interested in the activity context here. However, this is interlinked with the other areas of context. For example, two people might be meeting in an office or in a bar – obviously in both cases they have a meeting (or a discussion); however in the former case it is likely work related, while in the latter it is more likely to be of a social nature (especially if it is after work hours). So, the location and time lets us infer the nature of the activity. In the light of this, we cannot just look at a subsection of the context model, but rather require an overview of the whole model.

**The Context Model**

An overview of the proposed context model based on the considerations above and inspirations from literature is presented in Figure 3. The main inspiration of our context model is based on the ontology by Gu et al. [35], which provides a vocabulary for representing knowledge about a domain and for describing specific situations in a domain. However, their structure addresses taxonomies that are closely related with the more traditional physical sensors and hence provide a gap for our research. As we are dealing here with software sensors we have adapted some links to suit our needs more closely and expanded specific areas of the ontology. Other inspiration came from models such as the one targeting hardware sensors in CONON [41] or a specific domain in inContext [16].

The model defines a common vocabulary and structure of the context, allowing for sharing context related information between users and services to enable operations. It includes machine-processable definitions of the basic concepts in the domain and range and relations among them and reasoning becomes possible by explicitly definition. This model represents the main concept and subclasses that we require for our work in a convenient way – as always with ontologies one could debate little aspects here and there and we are not claiming that this model is superior to existing ones; it simply places focus on the elements that we require and presents these in a convenient way while being generic at the same time and thus allowing reuse in other domains and applications. Of course we are reusing ontologies where appropriate, such as for people (FOAF – Friend of a Friend) which is not detailed in Figure 3. To keep the figure readable, and to not overload the paper with detailed figures we have not included all possible links (For example all entities will have a location; some links might be created through attributes, others through reasoning rules) and also not all details of each class. We will now discuss each main class in a bit more detail and also hint at the kind of information that can be gleaned by combining data. The focus will be on the Activity ontology.

**Entity ontology (Person, Organization, Device)**

The entity ontology captures information about people, organisations and devices. Typical such information might include names and other descriptive data, but often also skills and capabilities – the latter showing what can be offered by the entity. For example an output device such as a screen would have resolutions and colour profiles as capabilities; a person might be a programmer capable of using specific technologies or languages. Most of this context information is semi-stable (it is less dynamic as learning new skills takes time and devices will not be constantly upgraded).
The social connection between entities must also be considered, and social connections are dynamic (people might have different relations to each other in different contexts). Devices might be (and predominately are) personal, but they might also share resources or accessible by a number of people. People might work for several organisations, e.g. in consultancy roles and worse there might not even be a very clear time separation between such activities as a person might be on the premise of company X while dealing with a call from company Y. Also, the relations between people are complex: a certain person might be the subordinate of another person in the company hierarchy, but for a specific project he might be the project leader and thus for the project outrank his superior.

Considering such relations and roles one could derive conclusions about activities: a lecturer in a room with 50 students on a University Campus is likely giving a class – if the room contains many fixed computer devices it might even be a computer lab and the class would be a laboratory class.
**Place ontology**

Places may have the names, are assigned to categories (e.g. restaurant), they will have physical locations (i.e. address, coordinates), and often associated resources (computers, tables, chairs, etc.) all providing useful context information.

The presence of an entity at different places lets us deduce much about their activities. At the most trivial level, people who are in their office are probably working while at home they are probably not working (unless we know that they are a mobile worker and the time of the observation tells us it is working hours). Of course the devices that they use and what they do with them influence this: someone using their mobile phone to undertake a call to their partner from the office is not working at that moment. In the context of a University environment, rooms have specific purposes and these let us deduce further information. A staff member alone in their office is probably preparing teaching material or undertaking research; while if they are in the office with a PhD student they are probably in a supervision meeting.

**Time ontology**

Events occur at specific times or might be scheduled to have start and end times. Sometimes time can only be provided as an interval enclosing the actual event time. A time instance is a point or an interval on the universal timeline [40].

Time is obviously often crucial when considering relations between facts – recall the example under the entity ontology where we loosely stated that the lecturer and the students are in the same room. Obviously a precise definition of that would need to refer to the time interval for the class and make precise that the students and lecturer are in the room at the same time.

Time plays a second very important role; not all context information will be immediately evaluated. Some might be stored for later use and here it is important to use the time to decide on the life time of an item of information. This is a significant problem in itself and is an item for future research. Consider for example a GPS coordinate for a user who is travelling in a car – this will very quickly be incorrect, however the city or country that he finds himself in will be more long-lived (unless we only identified the location just before he tried to cross a border).

**Activity ontology**

The Activity ontology considers two primary types of activities: scheduled and deduced. The former represents activities which are planned for a specific time and location with a specific purpose (e.g. a project meeting or a class or flight) and usually can be identified from calendar entries or teaching timetables or airline schedules. The later represents the activities that occurs in an more informal manner and can be inferred by combining context data through rules.

For example one might deduce that a user is missing his flight if he was scheduled to fly in 10 minutes, the airline flight information states flights are on time and the user is known to still be in his office away from the airport.

**Inference Procedures for Context Reasoning**

To derive new facts by applying rules on existing facts in the knowledge base is called inference mechanism. The most trivial inference is of course that where no items are combined (essentially a
lookup of some information). For example a scheduled activity can simply be looked up. Much more interesting is of course the use of several bits of information that together let us arrive at some new knowledge: knowledge gained by non-trivial inference. Such inference is guided by rules. For example we can deduce an activity by combining two or more items of context data, such as using a staff member’s work calendar, their role and location and the organization’s timetable we might derive that a certain Professor is indeed giving a class. We can argue that the more information we combine the better will be our judgement – in some way considering each piece of information as witnesses. Note that this does not come without downsides, as some information might be more reliable or up-to-date; we will return to this point later on when we briefly consider degrees of confidence in the scenario. Obviously we can store derived or inferred information back into the context model, on one hand to save the effort of re-computing it every time and on the other to ease its use in further deductions.

Simple inferences use the hierarchical structure of the ontologies; typically the sub and super-class relationships. The most common such form is transitivity ($A = B \land B = C \Rightarrow A = C$): if a person is physically present in Leicester and we know that Leicester is a city in the UK we can conclude that the person is in the UK.

More complex reasoning will involve domain specific rules that specify more intricate relationships between context items which cannot be directly observed from the ontology. However, these can be quite generic in their nature such as the impact of weather conditions on travel conditions and by extension the presence of people in certain places.

An example rule is (more examples are shown later when we consider an example scenario):

1. $\text{Timetable}(f,o) \land \text{Weather}(l) \Rightarrow \text{teachingClass}(af,c)$

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.

In general we could argue that if a person is currently in location X, is expected to be in location Y in 20 minutes and the normal time of transport between the two locations is 20 minutes, then the person should be able to fulfil the expectation. If we now also observe traffic delays or adverse weather conditions, we can deduce that the travel time will not be satisfied and hence the person will be delayed.

Considering the above rule, 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 (or at least make an announcement about the delay).

**Information Retrieval**

SPARQL[42] (SPARQL Protocol and RDF Query Language) is a query language which allows basic queries over RDF, such as conjunctive queries, UNIONs of queries (i.e. disjunction), OPTIONAL query parts, and filtering solutions by means of a set of basic filter expressions.

For instance, let us assume one wants to answer the following query: “Check whether a room is double-booked during a certain period of time, if so, get the list of all conflicted scheduled activities along with the attendants involved.” This could be expressed in SPARQL by means of a simple conjunctive query (assuming some technicalities, namely that we have "ctx" as default prefix.
http://www.cs.le.ac.uk/ontology/contextmodel# and that some OWL properties are defined in the ontology, such as hasVenue, hasTimespan, startsAt, finishesBy, supervisorOf):

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ctx: <http://www.cs.le.ac.uk/ontology/contextmodel#>


  ?act1 rdf:type ctx:ScheduledActivity.
  ?act2 rdf:type ctx:ScheduledActivity.

  ?act1 ctx:hasVenue ?loc.
  ?act2 ctx:hasVenue ?loc.

  ?loc rdf:type ct:Room

  ?act1 ctx:hasTimespan ?ts1.


  #check overlaps
  FILTER ((?start_time_act1>=?start_time_act2 &&
    ?start_time_act1<=?end_time_act2) ||
  (?start_time_act2>=?start_time_act1 &&
    ?start_time_act2<=?end_time_act1))
}
```

Context Acquisition

From the architecture (see Figure 1.) it is apparent that the sensors are placed in between the user and the services. These sensors capture the raw SOAP messages that are being used to invoke the web services, observing communications at the protocol level at the point where messages are reaching or leaving the Web Services Server. SOAP (simple object access protocol) [43] is an XML-based communication protocol to allow exchange of structured information between web services and applications over HTTP. SOAP is a textual protocol. By being payload to HTTP messages, SOAP messages are able to get around firewalls (a great advantage from practical points of view, but of course also huge concern to system security experts). However, SOAP is a W3C recommendation and the standard protocol for web service interactions.

Observing SOAP messages is quite simple, as handlers residing on both the client or/and the server side during a service invocation allow to intercept SOAP messages. These handlers can capture a SOAP request/response message before it is sent from or returned to the client. The handlers forward the information from the SOAP exchange to the sensors. The handlers are part of the standard SOAP server
architecture, albeit by default they are disabled for security reasons. The handlers can be activated for the client side (in the web.xml configuration file) and or for the server side (in the webservices.xml configuration file).

For example while creating a new entry in the Calendar Service, a user inputs data that will then travel from the client the web service using the SOAP protocol. These SOAP messages carry the arguments the user passes to create an event in the calendar, such as Title, FromDay, FromTime, ToDay, ToTime, Location and Description in the request. Figure 4 shows a typical SOAP message for the calendar service.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:ser="http://service.calendar.google.com">
  <soap:Header/>
  <soap:Body>
    <ser:getTitle>
      <!--Optional:-->
      <ser:title>Meeting</ser:title>
    </ser:getTitle>
    <ser:GetQuery/>
      <!--Optional:-->  
      <ser:fromDay>10/07/2012</ser:fromDay>
    </ser:GetQuery>
    <ser:GetQuery/>
      <!--Optional:-->  
      <ser:fromTime>18:00:00</ser:fromTime>
    </ser:GetQuery>
    <ser:getToDay>
      <!--Optional:-->
      <ser:toDay>10/07/2012</ser:toDay>
    </ser:getToDay>
    <ser:getToTime>
      <!--Optional:-->
      <ser:toTime>19:00:00</ser:toTime>
    </ser:getToTime>
    <ser:getLocation>
      <!--Optional:-->
      <ser:location>Leicester</ser:location>
    </ser:getLocation>
    <ser:getDescription>
      <!--Optional:-->
      <ser:description>This is a meeting with a colleague</ser:description>
    </ser:getDescription>
  </soap:Body>
</soap:Envelope>
```

**Figure 4:** An example SOAP Request message

We need to observe both requests and responses with our sensors, as they will carry different information. In the specific case here, the response will only be the acknowledgement stating that the entry is added into the calendar and thus not be very interesting in providing context information. For example for a weather service both messages will be important, as the request will specify a location and the response will inform about the actual weather.

**Mapping**

This section considers 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 (we discussed this earlier in the Background section), however at the concrete data level considered here we require novel techniques as we indicated earlier.

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 hypernym match). The fourth case is covering all those concepts that occur in terms and cannot meaningfully be matched to the ontology.

**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 5. 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.

![Figure 5: Overview of XML to OWL Mapping Methodology](image-url)
3. Find the tag in a lexical database (such as WordNet) providing a hierarchy (that is hypernyms). Match each hypernym 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.

**Direct Match**

The easiest kind of match is one where the tag matches the instance of the ontology (see also Figure 6), and in this case we wish to insert the value from the tag directly as literal associated to the instance:

![Figure 6: Direct Match](image)

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

**Synonym Match**

![Figure 7: Synonym Match](image)

The second type of match is where we find that a synonym of the tag (for example location and placement can be seen as synonymous in this context, see Figure 7) matches the instance in the ontology.

**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 \( t \), and
- \( \text{synonym}(t) \) be a set of synonyms for \( t \) as obtained from a lexical database then
$l = v \iff I \in \text{synonym}(t)$.

**Hypernym 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 and we consider the structural model to fulfil the needs of a context-aware systems making use of software sensors and hence we need to instantiate the model.

The proposed hypernym match (see Figure 8) is generic and could be used for any purpose where XML data needs to be inserted into fixed ontologies.

**Definition: Hypernym Match**

Let

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

then

$$l = v \iff I \in \text{hypernym}(t).$$

The hypernyms 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 hypernyms 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).

**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 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.

Sample Scenario

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.

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:

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ctx: <http://www.cs.le.ac.uk/ontology/contextmodel#>

SELECT ?faculty ?act WHERE{
  #all scheduled activities falls within the time span
  ?faculty ctx:hasActivity ?act.
  ?faculty ctx:supervisorOf ctx:a_student.
  ?act ctx:hasTimespan ?ts.
  FILTER ((?start_time>= "2011-22-02T9:00:00"^^xsd:dateTime
          && ?start_time<= "2011-22-02T11:00:00"^^xsd:dateTime)
           || ("2011-22-02T09:00:00"^^xsd:dateTime>=?start_time
              && "2011-22-02T09:00:00"^^xsd:dateTime<=?end_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.

**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 (we have seen rule 3 earlier):

1. \( \text{hasPlace}(?fm, ?pl) \land \text{Time}(?t) \rightarrow \text{isTeachingClass}(?fm, ?pc) \)
   
   where \( fm \) is a faculty member, \( pl \) is place name or name of a building, \( t \) is the time zone and \( pc \) is a postgraduate class.

2. \( \text{hasTimetable}(?fm, ?ocal) \land \text{hasCalendar}(?fm, ?pcal) \rightarrow \text{isTeachingClass}(?fm, ?pc) \)
   
   where \( fm \) is a faculty member, \( ocal \) is the organizational calendar (including a timetable), \( pcal \) is the personal calendar and \( pc \) is a postgraduate class.

3. \( \text{hasTimetable}(?fm, ?ocal) \land \text{Weather}(?loc) \rightarrow \text{isTeachingClass}(?afm, ?pc) \)
   
   where \( fm \) is a faculty member, \( ocal \) is the organizational calendar (timetable), \( loc \) is a location, \( afm \) is another faculty member (distinct from \( fm \)) and \( pc \) is a postgraduate class.

**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:

\[
\text{If (time is 8:00am and 5:30pm) and (day is Mon to Fri) then} \\
OC > WC > PC; \\
\text{Else} \\
PC > WC > OC
\]

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 the confidence of having correctly identified activities 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.

**Conclusion and Future Work**

In this chapter we have introduced software sensors as an inexpensive alternative to hardware sensors. They are particularly well suited for detecting user activities as they can exploit the multitude of data exchanged between users and the web enabled services they use. Hardware sensors will still have their place and are orthogonal to software sensors in that sense: they are very well suited to obtain environment data such as temperature or physical location measurements. The exchanged data enriches the information available about user activities and hence will allow for more effective context aware systems.

In the chapter we introduced the fields of sensors for context aware systems and context modelling. We presented our context model (an adaptation of ideas from literature that suits our needs well). We discuss how SOAP messages can be extracted from the exchanged network messages as a first stage in sensing through software sensors. We have presented a solution to the problem of associating data from SOAP messages with elements in the context model – thus allowing the generation of knowledge to be used in the context aware system. This technique 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 and is a revised version from that presented in [8].

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.

Two areas of study deserve more attention, which will be more important for software sensors, but also has implications for hardware sensors: (1) confidence in observations and (2) lifetime of information.

The former needs to study how much we can trust the information we have gained, that is how reliable is the sensor in reporting high quality information? This is very crucial, as reasoning about the data influence its reliability, and can do so by making it more reliable (two indicators showing the same fact are better than one), but can of course also degrade the quality (if we give high weight to bad information) – we are sure legal systems around the world cope with this problem, too! Of course the problem is
aggravated for software sensors (measuring e.g. the temperature using a hardware sensor is well understood), as we need to make decisions on data items and allocate them to specific categories.

The latter is also very interesting, as context data changes, but it does not do so at the same rate for all elements, and even for a specific element it might change at different rates: both a timetable and a calendar will provide insight into scheduled meetings; however the former is usually more stable mostly for practical reasons in that it is shared by more people.

The final issue for future work – and something we deliberately kept out of the scope of our current work is the matter of privacy. Obviously, any context aware system will violate privacy in some way by tracking what people do and while many people will feel that this is objectionable, there is a also a large number of people who make such information freely available on twitter or Facebook. Coming full circle, it is again location that is at forefront of the discussion in this area [44], but this discussion will need to be expanded in the future to cover activity sensing, too.

References
22


