# POSEiDON

PersOnalized Smart Environments to increase Inclusion of people with DOwn's syNdrome

## Deliverable D3.2

# Reasoning and Learning Module

| | |
|---|---|
| Call: | FP7-ICT-2013-10 |
| Objective: | ICT-2013.5.3 ICT for smart and personalised inclusion |
| Contractual delivery date: | M36 |
| Actual delivery date: | 31.10.2016 |
| Version: | V4 |
| Author: | Dean Kramer, Middlesex Univ. |
| Contributors: | Juan Augusto, Middlesex Univ. |
| Reviewers: | Silvia Rus, Fraunhofer<br>Sverre Morka, Tellu AS |
| Dissemination level: | PU |
| Number of pages: | 36 |

# Contents

## Executive summary

This deliverable documents our work on learning and reasoning within the POSEIDON project. In comparison to our deliverable "3.1 Ontology and Language" which considers representation capabilities, this document focuses on the potential for reasoning and learning. In the deliverable introduction, we revisit the aim of the deliverable as set out in the description of work.

Context reasoning and learning are introduced next. Literature in the field suggests rule based systems are the most popular reasoning technique.

The vision for reasoning within POSEIDON considers life outside the home, including resting, leisure, work, and education. The general system architecture is split across multiple components including mobile devices, and centralised services. This gives us the ability to carry out limited context reasoning on the device, and also more computationally intensive reasoning and learning off the device. In POSEIDON, we also aim to make the context acquisition self-aware in ways neglected in most context-aware systems. This essentially encompasses enabling the system to decide how to adjust its context acquisition based on the current context of the device.

To validate our architecture we define two basic types of assistance: *reactive,* and *proactive*. Reactive situations are those in which particular events happen and the user must take action. This contrasts proactive situations where the system 'looks ahead' to assist the user to avoid undesirable circumstances, for example picking the correct clothing to suit the weather in the location s/he is going that day. These different types of scenarios are driven by results of the deliverable "D2.1 Report on Requirements". Privacy is an additional area to which reasoning and learning has been applied, enabling a more dynamic and adaptable privacy policy.

In POSEIDON, we have developed a number of different technologies to help support context-aware systems development. This firstly includes a modelling notation designed to model the different contexts to be used in the particular system, and their inference rules. Secondly, we have developed an opensource context middleware, which reasons over raw context data and existing contexts in POSEIDON mobile applications. This middleware uses the context library we have developed earlier, and uses the C-SPARQL reasoning engine. It is developed as an Android application, which is now on the Google Play Store. Additionally, we created a learning module which the primary and secondary user can use to analyse the evolution of relevant behavioural traits over different time periods.

# 1. Introduction

The aim of this deliverable as stated in the DOW is:

> *D3.2) Learning and reasoning module: software providing learning and reasoning capabilities in relation to contexts. There will be several releases as the system evolves and different validation exercises are used to inform the next stages of development. More fundamental functionality will be considered first and successive refinements will add less critical services. Interim reports will be delivered months 10, 20 and 30.*

Context-aware applications have been described *to be intelligent applications that can monitor the users context and, in case of changes in this context, consequently adapt their behaviour in order to satisfy the users current needs or anticipate the users intentions* (Daniele *et al.*, 2009). To enable the system to such tasks, it will need to reason over the different contexts of the user, devices, and environment. In this deliverable, we aim to introduce and describe how this is carried out in POSEIDON. Over the project, we have enabled this behaviour through the development of developer tools, including a modelling notation for context inference rule creation and a software library to support component creation, and end-users through the development of a centralised mobile reasoner, and a learning module to support the primary and secondary user in understanding past events.

The remainder of this document is broken down into the following sections: Section 2 surveys related work in the areas of context reasoning and learning. We then introduce the current vision for reasoning and learning in Section 3. Next in Section 4, we describe the reasoning modelling notation, designed to support context modelling and translation to our chosen language in D3.1. In Section 5, we introduce and describe the developed context middleware for the acquisition and reasoning of a context. Section 6 introduces our learning module designed to assist primary and secondary users. Finally, Section 7 concludes this deliverable.

# 2. A Survey on Context Reasoning and Learning

In this section, we shall introduce related work in the areas of context reasoning and learning.

Context-Awareness is a relatively established area within Computer Science. Our deliverable “D3.1 Ontology and Language” provided a summary and an initial evaluation of some of the possibilities available to this research project. The emphasis of that deliverable was more on the representation capabilities, whereas this deliverable builds on that initial survey and goes deeper focusing on the potential for reasoning and learning.

Knowledge representation is linked to reasoning. However there is no guarantee that a tool which is good to represent knowledge is necessarily good for reasoning, hence the rationale for this independent analysis of reasoning and learning. This explains why some options which were considered at that stage are not developed here, as the potential for reasoning and learning are superseded by those included in this report. Notice this is not to say that options considered in D3.1 which are not listed are not valuable to us, but instead that they may be used as an entry point for information which later on is translated to one of the tools we consider in this report. This integrated view of the different components of the context-aware part of the project is at this stage being discussed with a few combinations under consideration.

Reasoning systems considering a notion of ‘context’ have been proposed in traditional AI (Benerecetti *et al.*, 2000), however these consider reasoning at a level of complexity and generality which is not required nor affordable for the type of system and devices considered in this project. Those lines of research have also remained inactive since discussed in the ‘90s and there are no records which show

their success at implementation level. Other traditional reasoning methods have been adapted and used in our area in the last decade. Lim and Dey (2010) provided recently a good summary which is summarised in Figure 1. It shows the predominance of rule based systems for reasoning and the adoption of a diversity of classifiers for context recognition. Sandri (2011) provides a more detailed analysis of the rule-based approaches. An important advantage of using rule-based systems is that they are more amenable to forma verification of correctness (Augusto, 2005; Augusto and Hornos, 2013). This is an important feature in any system, especially in AAL systems, on which human beings may depend upon.

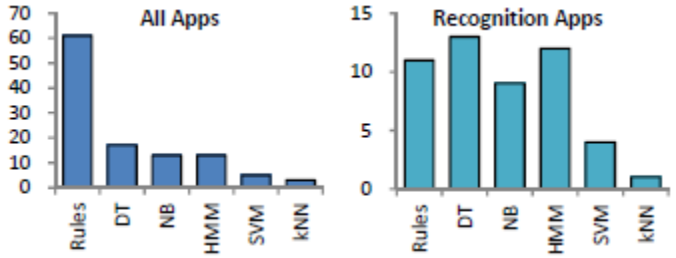



**Figure 1 Most popular reasoning techniques and classifiers user for context-awareness as presented in** Lim and Dey (2010)**. Key:** decision tree (DT), naïve Bayes (NB), hidden Markov models (HMM), support vector machines (SVM), k-Nearest Neighbour

As it can be observed the decision of what reasoning mechanism to use opens many possibilities. Our project also has specific constraints in terms of the computational resources required and the reliability expected from the inferences which makes this decision even more difficult. The subsequent sections mention other alternatives which have been created outside the Artificial Intelligence area.

## 2.1 The Context Modelling Language (CML)

CML allows the developer to model context information and requirements graphically (Henricksen *et al.*, 2002; Henricksen and Indulska, 2004, 2006). CML is formulated using concepts from Object-Role Modelling (ORM). This provides a relational database query based type of framework with a closed world assumption. The representation of tuples is interpreted in such a way that it has an associated semantics of a three-valued logic (true/possibly true/false). A concept of situations is created out of lower level contextual information. Situations are handled through a First Order Logic (FOL) with restricted quantification. An example situation specification is show in Figure 2.

$$Occupied(person):$$
$$\exists t_1, t_2, activity \bullet \texttt{engaged\_in}[person, activity, t_1, t_2] \bullet$$
$$(t_1 \leq timenow() \wedge (timenow() \leq t_2 \vee isnull(t_2)) \vee$$
$$(t_1 \leq timenow() \vee isnull(t_1)) \wedge timenow() \leq t_2) \wedge$$
$$(activity = \text{“in meeting”} \vee activity = \text{“taking call”})$$

$$CanUseChannel(person, channel):$$
$$\forall device \bullet \texttt{requires\_device}[channel, device]$$
$$\bullet \texttt{located\_near}[person, device] \wedge \texttt{permitted\_to\_use}[person, device]$$

$$SynchronousMode(channel):$$
$$\forall mode \bullet \texttt{has\_mode}[channel, mode] \bullet \texttt{synchronous}[mode]$$

$$Urgent(priority):$$
$$priority = \text{“high”}$$

**Figure 2 Example of a situation specification (occupied)** (Henricksen and Indulska, 2006)

This concept is supplemented with a system of preferences, triggering situations in an Event-Condition-Action (ECA) rule fashion (upon-when-do). Figure 3 below shows the overall structure of the system.

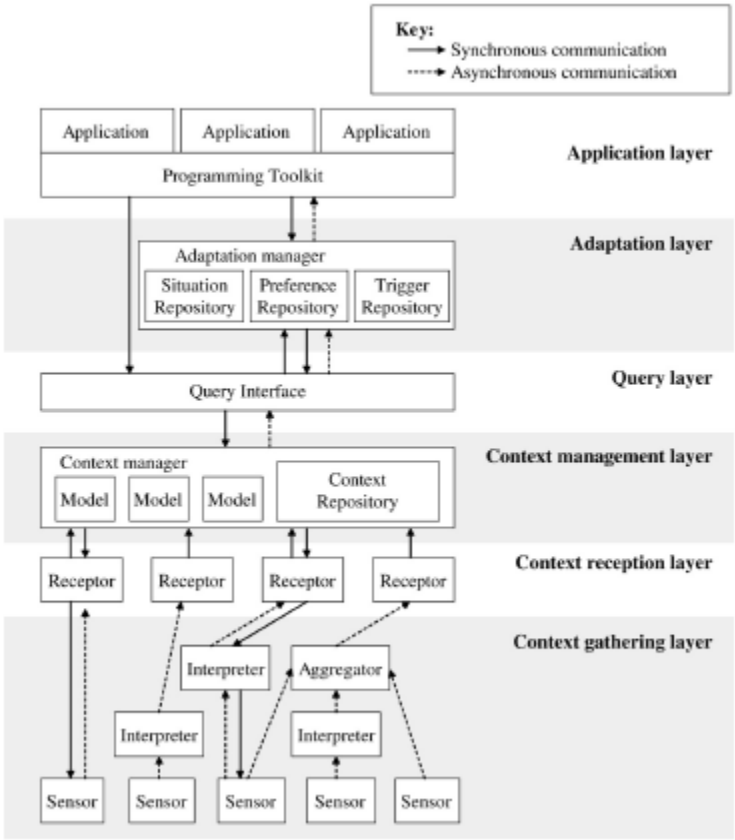



**Figure 3 Example of a situation specification (occupied)** (Henricksen and Indulska, 2006)

CML has some limitations like the lack of capacity to structure knowledge or to reason with different categories as all contexts are at the same level. There are also limitations in the domains that can be covered.

## 2.2 Other Context Modelling Approaches

Historically, context models and languages have predominantly been either *key-value models, markup scheme models, graphical models, object-oriented models, logic based models, or ontology based models* (Strang and Linnhoff-Popien, 2004).

Key-value models are the simplest form of context models, involving a name and context value pairs, which have been used directly (without the use of AI techniques applied) in context-aware systems and frameworks (Kramer *et al.*, 2011). Markup models are hierarchical data structures which consist of markup tags, attributes, and content, with an example being the Comprehensive Structured Context Profiles (CSCP) (Held *et al.*, 2002). CSCP is based on Resource Description Framework (RDF), and expresses context information using service profiles, describing context information relevant to different sessions. Graphical models can use different graphical notations including Unified Modelling Language (UML), Object-Role Modelling (ORM), and other domain specific modelling languages. ORM based context models include the Context Modelling Language (CML) (Henricksen and Indulska, 2006). UML based models include ContextUML (Sheng and Benatallah, 2005), and the MUSIC context model (Reichle *et al.*, 2008). These models can also be viewed as object-oriented models, as they use different object-orientation concepts including inheritance, and encapsulation. Other domain specific languages (DSL) for context modelling include PervML (Muñoz *et al.*, 2004; Serral *et al.*, 2008) and MLContext (Hoyos *et al.*, 2013). These DSLs take a given context model and generate code for use in the final system, in the case of PervML a set of Java classes and OWL specifications, and MLContext, Java classes for the Java Context-Aware Framework (JCAF).

Many different logic based modelling languages also exist, including the Calculus of Context-Aware Ambients (CCA) (Siewe *et al.*, 2011), CONAWA (Kjærgaard and Bunde-pedersen, 2006), SCAFOS (Katsiri *et al.*, 2010), and an algebra of contextualised ontologies (Cafezeiro *et al.*, 2008). The CCA proposes a logical language for expressing context properties using context expressions. Context expressions can be composed to form complex expressions and formulas using first order operators. The CONAWA calculus was inspired by the ambient calculus (Cardelli, 1999), and extends it in a number of ways. First, it extends the syntax with constructs and capabilities allowing ambients navigation in complex context information. Second, it extends the semantics of different ambient capabilities to handle contexts, and context trees. The algebra of contextualised ontologies aims for a uniform representation of entities & context, and places an emphasis on the relationships between them. Different modular constructs are proposed to be applied to contextualised entities to combine entities and contexts coherently. These constructs are broken down into three classes including: Entity Integration, Context Integration, and Combined Integration.

Using ontology based context models can bring a number of different benefits including knowledge sharing, logic inference, and knowledge reuse (Wang *et al.*, 2004). User modelling ontologies include the General User Model Ontology (GUMO) (Heckmann *et al.*, 2005), and User Navigation Ontology (UNO) (Kikiras *et al.*, 2006). GUMO provides different dimensions of the user to be modelled, including characteristics, emotional state, personality, and physiological state. UNO on the other hand extends GUMO with concepts including mental ability, mobility ability, sensory ability, spatial ability, demographics, and preferences. Ontologies for ambient intelligence include GAIA (Ranganathan *et al.*, 2003), CoDAMoS (Preuveneers and Berbers, 2005), OntoAMI (Santofimia *et al.*, 2009), and BOnSAI (Stavropoulos *et al.*, 2012). GAIA includes physical (stock quotes, sport scores), personal (health, mood,

schedule, activity), social (group activity, social relationships), application (email, websites visited), and system (network traffic, status of printers) concepts. The CoDAMoS ontology on the other hand considers the user, platform, service and environment. This ontology is also directly extended in BOnSAI to include context-related, service-related, hardware related, and functionality-related concepts.

Ontologies for Ambient Assisted Living (AAL) systems supporting elderly people have been proposed (Zografistou, 2012). This collection of ontologies included a core ontology, a person profile ontology, a health ontology, and a time ontology. The core ontology is used for general purpose concepts including location, environment, simple events, person, activity etc. The person ontology added the ability to model the person's status, habits, impairments, contact profile, and preferences. The health ontology included the concepts disease, symptoms, treatments, and restrictions. Other health related ontologies includes OntoHealth, an ontology for pervasive hospitals (Librelotto *et al.*, 2010). This ontology included concepts for patient, bed, tray, nurse, medical history, dosage, physician, exam, medicine, and treatment. Other general purpose context model ontologies have been proposed including SOUPA (Chen *et al.*, 2004). SOUPA was built using a collection of reference ontology vocabularies including FOAF, DAML-Time and the Entry Sub-ontology of Time, OpenCyc, Regional Connection Calculus (RCC), COBRA-ONT, MoGATU BDI ontology, and the Rei policy ontology. This ontology is broken down into two distinct ontologies; SOUPA Core for generic pervasive applications, and SOUPA Extension for specific pervasive domains.

## 2.3 Ontology-based Reasoning

While we have introduced related work leveraging ontologies for modelling context, these models can also be used for reasoning and logic inference. Particularly, if we look at the most established ontological frameworks, the Web Ontology Language (OWL), we can use Description Logic (DL) as an approach for reasoning. We have mentioned a good number of ontology based research in deliverable D3.1. Some of the issues associated with the use of DL as the reasoning engine include problems with complexity and consequently scalability.

## 2.4 Situation-based Reasoning

Some research has based their representation on the notion of "situations". This term is used to provide a hierarchy of concepts which extends above simpler, low-level contextual information or cues collected from sensors. Dey (2001) defined context as: "*any information that can be used to characterise the situation of an entity*" where an entity can be a person, place or object considered relevant to user and application, including the user and the application themselves. Dey also defined situation as: "*description of the states of relevant entities*", which makes them temporal.

There has been a long tradition of reasoning with situations in AI, the most well-known being the Situation Calculus (Reiter, 2001) which is based on a second-order logical language for reasoning about action and change. Although expressive, the notion of situation in the Situation Calculus is different and the computational complexity of reasoning with these concepts is high. This means that it will be difficult to use them in practical situations which are assessed in machines with very limited computational resources like a mobile phone.

Reignier *et al.* (2007) represented situations as a Directed Graph labelled with temporal relations as in Allen's Temporal Logic (Allen and Ferguson, 1994) which can then be automatically translated into a Petri Net. One problem is that Petri Nets are difficult to handle when the situation has many parts and also to consider a big cluster of situations with several connection points.

## 2.5 Reasoning with Uncertainty

Many different types of uncertainty based reasoning mechanisms have been proposed in the past, including Dargie (2007), who proposed a conceptual architecture with a mix of different reasoning mechanisms: fuzzy logic to define the conceptual states of a primitive context, Dempster-Shafer Theory to combine the independent observations of multiple sensors and Hidden Markov Models and Bayesian Networks to compute higher-level contexts. Augusto *et al.* (2008) proposed a rule-based system of an Event Condition Action (ECA) style enriched with uncertainty and temporality. However these systems all pose the challenge of maintenance as uncertain information is not static and their values change through time, sometimes quite dramatically and also quite often. The question is, how can we continuously update this information and how do we know they are reasonably accurate representations?

## 2.6 RDF/OWL and SPARQL

There has been some work recently on the use of RDF as starting point to represent knowledge in Ambient Intelligence systems, which provides a representation which can be fed to different reasoners. One popular language to reason about RDF data is SPARQL which is followed as a standard query language for RDF by W3C, an example being Gueffaz *et al.* (2012). Other researchers have considered OWL as a departing point, see for example Meditskos *et al.* (2013).

These options have their own intrinsic limitations in terms of expressiveness of RDF, OWL, SPARQL and DL (frequently associated as a reasoned for OWL), especially compared with traditional AI based reasoners. However it has some practical advantages and it may provide a starting point on top of which to develop other capabilities on demand and as far as allowed by the capabilities of the computational devices we use.

## 2.7 Context Learning

Whilst we clearly perceive the benefits of having learning capabilities informing the system, hence our inclusion of a learning module in the proposal, we believe that the development has to focus on reasoning first and learning later. Focus on reasoning will help us clarify what services are most important. Then, we will investigate how learning can inform reasoning about those specific elements.

As for any intelligent system, learning provides substantial support to reasoning and Ambient Intelligence is no exception (Aztiria *et al.*, 2010). There has been interesting advances in learning habits of users from activities perceived through sensors, see for example: Muhlenbrock *et al.* (2004) and Aztiria *et al.* (2013). This learning process is usually offline and it considers data which have been accumulated through several weeks or months. For example, a person living in a smart home performs several activities of daily living each day and the sensors that are triggered when the person is moving in the house and using several appliances are stored in a database. Then an algorithm is given these databases as an input and the goal of the algorithm is to find common patterns of behaviour, routines and activities which are performed in a reasonably regular way, to the extent that the system can anticipate them and do something to facilitate some of the daily living chores. However in our project, we support a person mostly outside home. The only sensors most likely to collect their activities and preferences are those in the mobile components (e.g., phone or tablet) hence the learning objectives and the process for doing it may be different.

# 3. Our Vision for Reasoning and Learning within POSEIDON

In this section, we introduce and discuss the current vision for the use of learning and reasoning in the POSEIDON project. First we consider the role of AAL and context-awareness.

Our project focuses on the task of bringing some of the latest technological advances to increase inclusion in our society of a specific group of citizens: people with Down's Syndrome (DS). The overall ethos of the project is to focus on the best abilities of people with DS and to help them to live their lives in a more fulfilling way by facilitating access to education, work and social events. The system is currently under development and it can be largely identified with what we call Intelligent Environments (Augusto *et al.*, 2013) or more specifically AAL systems (Augusto *et al.*, 2012). Artificial Intelligence (Russell and Norvig, 2003) has been considered as a technological area which can make a substantial difference in the quality of services offered by AAL systems through *ambient intelligence* (Augusto, 2007; Ramos *et al.*, 2008). Although a substantial difference is that whilst AAL systems focus more on supporting life indoors, through the realisation of e.g. Smart Homes, POSEIDON puts more emphasis on life outside the home. People with Down's syndrome often face the challenge of integration in our society and our project aims at supporting them to be more positively immersed in society. An essential step to facilitate their integration is to help them reach, as independently as possible, the places where they can learn, develop a profession, and socialise.

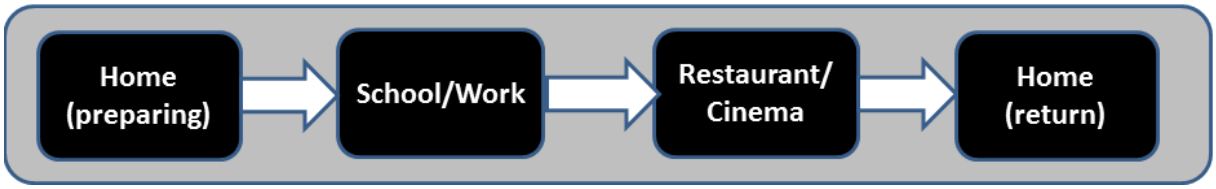



Figure 4 Main supported scenarios

Figure 4 shows an illustration of the type of scenarios we consider in the project. Imagine a person with Down's Syndrome leaving home in the morning to go to work. There are challenges on navigating through a busy city and there may be unexpected situations which further complicate the journey. At all times the person has to feel secure and also their family wants to feel reassured that everything went well.

All these systems rely heavily on a module providing context-awareness and this is the focus of this deliverable. Context-aware applications have been described "…*to be intelligent applications that can monitor the users context and, in case of changes in this context, consequently adapt their behaviour in order to satisfy the users current needs or anticipate the users intentions*…" (Daniele, 2006).

## 3.1 General System Architecture:

The system to handle the Ambient Intelligence and context-awareness is split across multiple components:

- **Mobile Devices**: These devices are used by the person with Down's syndrome to assist them when they are away from home. These will contain the main context-awareness and reasoning system. It will ensure that in conditions of limited network connectivity, it will still function
- **Centralised Services**: These services contain the majority of the learning systems. It will collect contextual information from mobile devices, and be able to provide visualisations to the users on behavioural traits.

**Context Acquisition Self-Awareness:** While the system can be adapted to suit a given context, so can the context acquisition system itself too. Context acquisition frameworks and systems historically have been static entities that only adapt the end application, not themselves (Fonteles *et al.*, 2013). By making the context acquisition system self-aware and adaptive, the system can better optimise itself in a changing environment and condition. As an example, let us consider location-aware services.

Depending on the current battery status, it can be beneficial to alter the frequency in which the device attempts to locate itself. This can allow the system to preserve power, when the battery is running low.

Other uses of self-adaptive context acquisition systems include switching between sensors in different settings. For example, while Global Positioning System (GPS) sensors can provide location information outside, they are of little use inside. Instead, for indoor location tracking, we can use Wi-Fi positioning (di Flora and Hermersdorf, 2008). Therefore mainly only one approach needs to be used at a given time, and should change automatically. Although in transition areas (very close to the home or work) more than one may be available, which is also a useful context information.

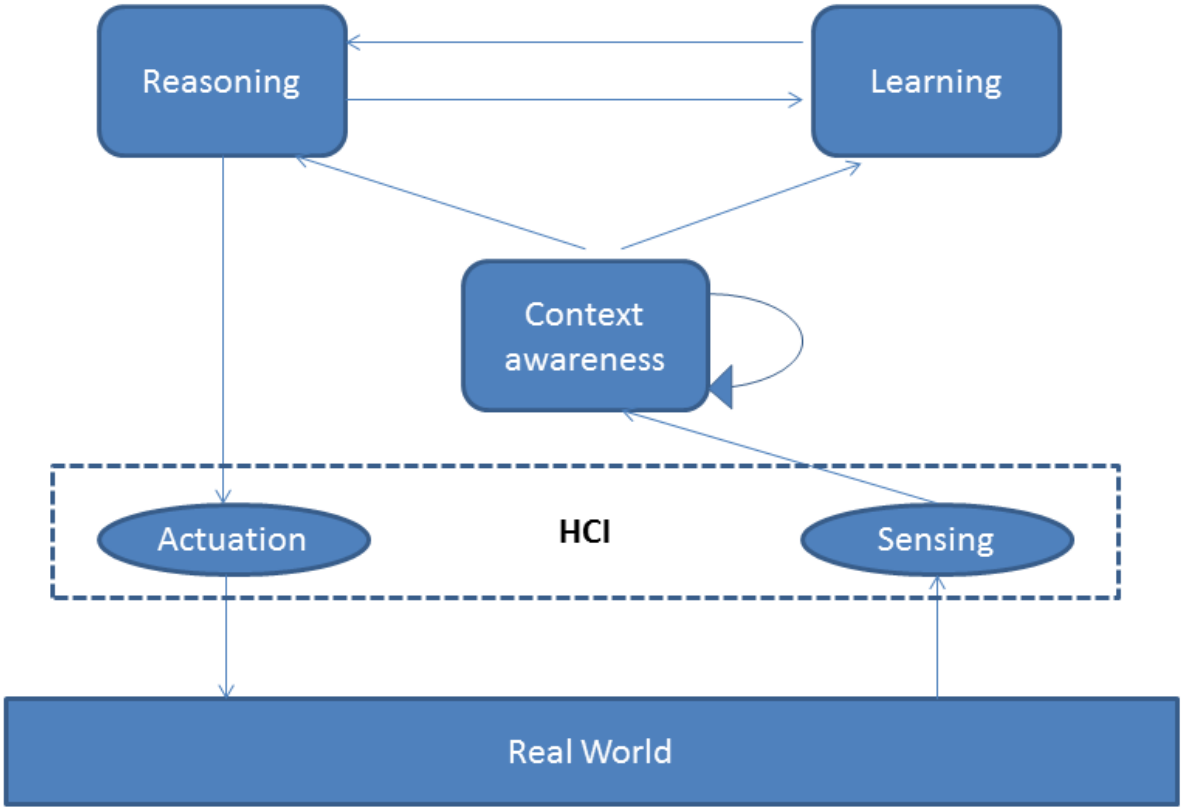



Figure 5 Context-aware focused sub-system architecture

In Figure 5, we illustrate the system architecture. It is only a partial view of the POSEIDON system and it is mainly focused on the modules which we are discussing in this deliverable: the interplay between Context-awareness, Reasoning and Learning.


Figure 6 Overall system architecture

Figure 6 provides a more general overview of the technologies which make up the POSEIDON system. For example, cloud services are used to retrieve different type of information (e.g. GIS) which informs the context awareness module. Although this is not the focus in this deliverable we believe this contributes to the understanding of the topics of this article in the broader system.

## 3.2 Validation Scenarios

Our project has spent significant resources interviewing primary users (i.e., people with Down's Syndrome) and secondary users (i.e., those who regularly help support them, typically a relative or formal carer). This analysis included more than 300 online questionnaires answered in Europe, more than 20 interviews, workshops with families (including a person with Down's Syndrome in each). This gathering of data allowed us to shape the requirements for the project. The results of this endeavour can be found in our deliverable "D2.1 Report on Requirements". This has further given us user personas which have been used within our validation scenarios. Using the results of this task, we were able to characterise our assistance to two specific types: *reactive* and *proactive.*

Below we offer examples of these and we provide a generic comment on how the different modules of the architecture presented in Figure 5 are exercised.

**Reactive Scenario:** As a reactive scenario, we consider a user travelling to work.

- World: Currently walking to work in the morning.
- Sensor: Coordinates the device and collects the current time.
- CA: Associates that the user is continuously deviating from the route
- Reasoning box:
    - User is making many smaller deviations continuously.
    - User may be unable to find the particular building they need to go.
    - User may require assistance from Carer
- Actuation: Offers the user to contact their Carer for support.
- Real world: Message is delivered

**Proactive Scenario:** As a proactive scenario, we consider a user walking to meet a friend in the city.

- Sensor: Finds the user has an appointment with a friend in his calendar.
- CA: Checks the weather conditions for the area associated with the appointment.
- Reasoning box:
  - Finds the weather at the listed location is bad, e.g. raining.
  - Suggests that the user should wear appropriate clothing, and reminds the user to be careful with the device outside to avoid damaging it.
  - Prompts the user if they still wish to walk, or if they would rather take public transport, and awaits response.
- Learning: Stores that when the weather is bad, the user prefers to either walk or take public transport.

These scenarios represent some examples on how POSEIDON can interact and react with the different contexts. In addition, these scenarios were presented at the workshop on artificial intelligence applied to assistive technologies and smart environments at the 28th AAAI conference (Kramer *et al.*, 2014).

## 3.3 Other Tentative Learning and Reasoning Scenarios

Continuing from the scenarios described above, here we discuss other areas in which the learning and reasoning can be applied.

As the different software context-aware and learning components will require data collection, user privacy is an important issue that needs addressing. This privacy currently is most aimed at the ability for the secondary user to access their primary user's data including their current location. In the POSEIDON project, we aim to give the users the ability to take control of their data privacy. Over POSEIDON, we considered the three following policies:

1. **No Privacy:** This preference allows the secondary user to easily get the location of the user without prompt, or consent each time.
2. **Consent Required:** This preference requires the explicit consent of the primary user each time a data request is sent by the secondary user.
3. **No data:** This preference not only does not fetch any data back to the secondary user, but it also does not prompt the primary user.

These policies provides some levels of privacy for the user, but as situations change, perhaps in a potentially concerning way, the privacy settings may also change as well.

To address this, we believe that standard static privacy policies may not be an optimal solution. For example, in the situation of a primary user being in danger, a static privacy policy could prevent the parent, carer, or law enforcement from being able to locate that user. This leads us to add the option of dynamically overriding the privacy settings if there are reasons to be concerned.

One example of this includes using location and calendar based contexts. If the primary user is expected to be in school between specific times, yet they are monitored to be moving away from the school, this could be an event to inform the secondary user. A second example includes learning the user's mobile phone habits, for example how many times the primary user calls their secondary user. If no phone activity is found over a prolonged period of time (over 6 hours), and the user is not in a location destined as a safe location, this could be perceived as a dangerous situation. In both these examples, the application can be designed to lower the level of information privacy with the secondary

user, to help locate the user in these exceptional situations. This reasoning can be carried out both on the mobile device and central systems based on different needs for example, minimal network usage, or battery usage.

# 4. Reasoning Models

Back in Deliverable 3.1 Ontology and Language, we introduced C-SPARQL as our language of choice for carrying out atomic level context inferences. To generate some form of stream reasoning based on languages like C-SPARQL, different queries often need to be created by hand. Creating these queries however can often be error prone, and also include a lot of boilerplate which could be present in many queries. This is particularly the case when contexts can contain many different states. Understanding a context tree hierarchy can be made more difficult if all rules and queries are based purely in text. For this reason, we propose a modelling notation to be used for modelling the reasoning rules of a context-aware system, which can then be translated into C-SPARQL rules. In Figure 7, we depict an example model consisting of rules for the battery, and if the user requires navigation assistance.

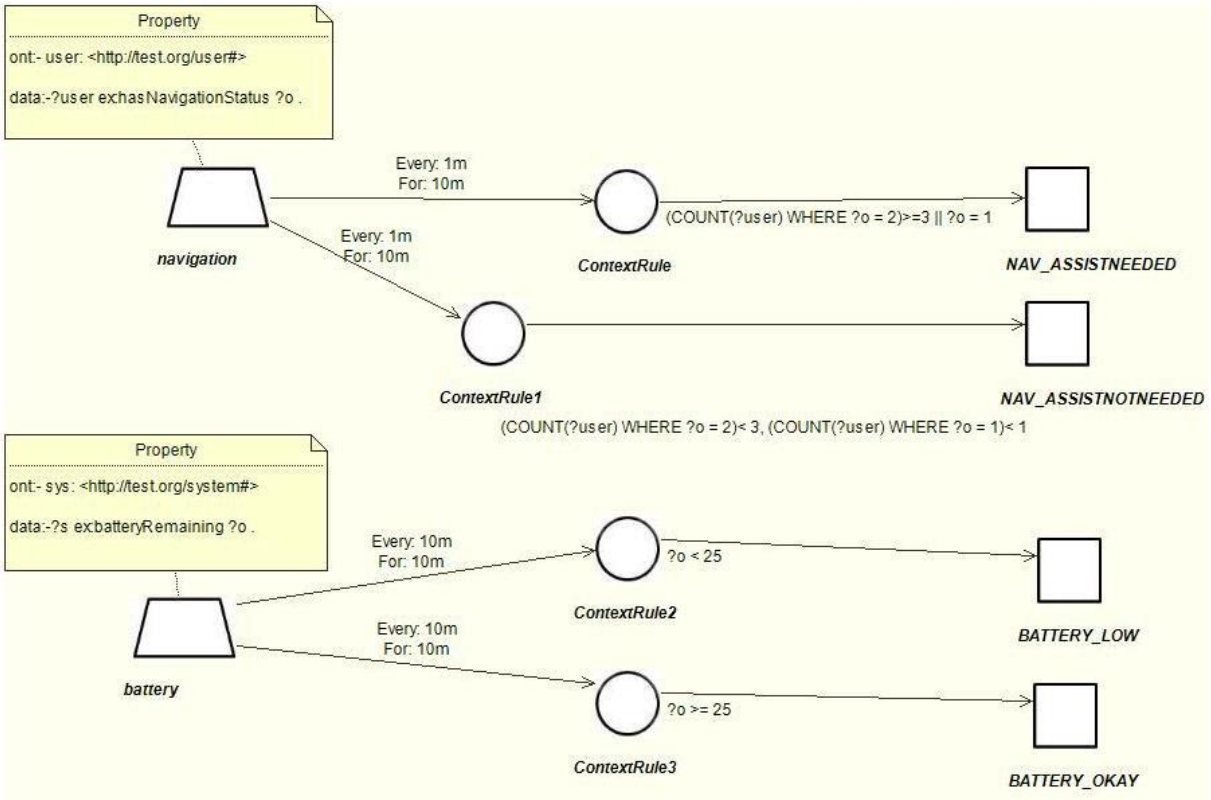



Figure 7 Context Reasoning Model

## 4.1 Model Constructs

In our modelling notation, the models are built through the use and interaction of different model constructs, this includes *Data Sources, Inference Rules*, and *Context States*. In Figure 8, we show a key for the different construct representations in the models.

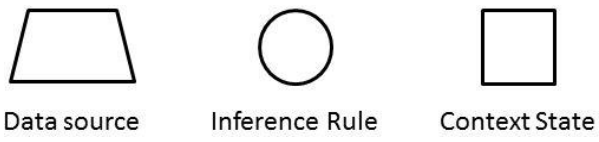



Figure 8 Model Constructs

### 4.1.1 Data Source

What separates context-aware systems from regular systems is the ability to infer the context of the device, user, and environment. For the developer to express rules to carry this out, there needs to be raw data for which can be reasoned over. This raw data can come from a variety of sources include sensors, and other off device services including web services. In the modelling notation, a context source object denotes a particular source of raw context data. This raw data is expected to be received in RDF triples. In the model, context sources are denoted with trapezium objects. Each context source can interact with more than a single inference rule. For each context source, there are different properties that can be added. The first includes an *Information Source*, which is used to define specific SPARQL ontological prefixes. The next property is *Data*, which relates to the RDF triples that the implemented context source provides. Just as in SPARQL, we just give the variable a name with a question mark (?) prefix. These variables can then be used in the Inference rules. A single context source can have a relationship between many different inference rules.

### 4.1.2 Inference Rule

Raw data alone is not always ideal for creating context-aware applications. For example, when considering remaining battery life in a mobile device, often people are interested to know when it is low, not if it is 67% charged. Having defined the raw data expected to be received by the device, to create more meaningful atomic context information, this raw data needs to be used in a context inference. This object denotes different inference rules that are used to infer a particular context state. Inference rules in our modelling notation can contain a mix of both simple *logical evaluations*, and more complex *functions*. Logical evaluations in the inference rules allow the developer to create rules based on a particular logical conditions, for example ``if battery level is greater than 25%''. To use logical expressions, the developer can reference particular variables from the context source. Only variables from a context source related to that particular context rule can be used. For multiple logical expressions, these can be separated with a comma allowing a conjuncture of different expressions.

Function usage within an inference rule are related to the different functions built-in to C-SPARQL/SPARQL. These functions can allow the developer to do far more computations over the raw data to assist inference. An example from our project includes calculating the total sum that a person has walked in a particular interval. In the inference rule, these functions can be used whereby the developer states the function signature, any variables required in the parameter parentheses. If the developer wants a particular set of RDF triples being used with the functions, they can be stated. By default, these functions are called with all RDF triples known the inference rule itself. Lastly, the developer can also apply logical evaluations to the returned result of those functions.

### 4.1.3 Context State

Having defined the particular inferences the system needs to make on the raw context data, the developer can use context states to be applied if the inference rule evaluates as expressed. Context states can be used for both single atomic contexts, and higher level aggregation contexts.

## 4.2 Relationships

In addition to these context constructs, there are relationships that exist between them. These construct relationships include:

- **Source – Rule:** This type of relationships defines what raw data the inference rule queries over. Included in this relationship is temporal operators regarding the raw RDF, including *data window size*, and *execution frequency*. Data window size relates primarily to how much RDF data can be included within the rule, and is noted by `For`. As featured in C-SPARQL, the data

window can be either *physical* (a given number of RDF triples) or *logical* (a triples occurring with a given time interval). Execution frequency relates to how often the inference rule is run in the reasoner, and is noted by `Every`. This frequency corresponds to sliding logical windows (Golab and Ozsu, 2003) in C-SPARQL, and in essence refers to the possibility of RDF triples being present in more than a single logical window. In practice this will equate to how often that query is updated. If the data window size is physical with exact triples being set, this field is not required.

- **Rule – State:** Inference rules themselves do not include the specific context states the rule equates to. This separation allows the developer to connect more than a single rule to a context state, which can encourage reuse.
- **State – State:** So far, we have considered the modelling of lower level, atomic contexts. These contexts deal with singular situations of an element, based on raw data from sensors. When developing context-aware systems, the ability to aggregate, and reason over a collection of different contexts gives the ability to consider higher level knowledge on a situation. For this, we allow the user to aggregate contexts together in our notation. To aggregate context values, we join the different context states together, and add a logical operator (*AND, OR, XOR*). If no logical operator is given explicitly, the default value is a conjunction.

## 4.3 Rule Generation

Following the construction of the overall context model, we need to generate the C-SPARQL queries and aggregation rules to be deployed on the mobile device. Firstly, we get a list of all atomic, and aggregate contexts in the model based on object relationship. For atomic rules, context states will need to contain relationships with inference rules. In comparison, aggregation rules will purely have relationships with just other context states.

A C-SPARQL query is created for every atomic context. To do this, first the tree of objects that relate to that atomic context are retrieved. This includes inference rules, and context sources that are linked to those inference rules. Queries names are generated using a concatenation of both the context state and `_query`. Therefore a context state such as `NAV_ASSISTNOTNEEDED` will create a query named `NAV_ASSISTNOTNEEDED_query`. Next, query prefixes are generated, by using `ont` properties of the related context sources objects. Following this, the type of C-SPARQL query is declared, in our case we use `Construct` queries, which return an RDF graph based on the result of the query. Next, we set the temporal operators of the C-SPARQL rules getting the execution frequency and data window size from the source-rule relationship. In the event there are multiple context sources with different temporal settings, we set the frequency to be same as the most frequent using a data window size as the same as the largest. Using these two temporal settings, we use the data window size value for the `Range`, and execution frequency for the `Step` parameters. Next, `Where` clauses of the query are generated. Firstly, we copy all the data values from the related context sources in the rule object tree including all declared variables in RDF. Following this, we generate any subqueries that might be required in the context query. Subqueries within our modelling notation are generated only for handling functions within C-SPARQL. This generation is explained in the following subsection. Finally, we generate SPARQL `Filter` clauses for any inference rule logical evaluations. If multiple logical evaluations have been used, each comma separated, a filter is created for each one.

### 4.3.1 Handling Functions

As introduced earlier, we allow the use of SPARQL aggregate functions in inference rules. When generating C-SPARQL rules, these are handled through the generation of subqueries. For every function used in the inference rule, a `Select` subquery is generated. These generated subqueries are

places within the `Where` clause block in the main context query. In this generated select query, we first get the function name and any passed parameters. Next we have to give the function return variable a name which is added after the `AS` keyword, which for this generate a name `subqres_` with an incremented number. An example can be `COUNT(?user) As ?subqres_1`, which would counts the number of occurrences of the value held in variable `?user`, placing the resulting number in the `?subqres_1`. Following this, we create the Where clause copying the raw data RDF from the context sources, the same as the main query.
Finally, where functions usage have a logical evaluations to be used over the resultant variables, we generate Filter clauses for each evaluation using the variable generated automatically earlier.

### 4.3.2 Aggregate Contexts

As described earlier, we do not use C-SPARQL for context aggregation. Instead, we generate propositional formulas to represent our aggregation rules. These rules are evaluated on the deployed mobile device by means of boolean satisfiability. For satisfiability tests, we generate propositional rules that use equality as a method of determining if the aggregate context is true. To generate these aggregation rules, we check the model for Context State-to-Context State relationships. These relationships occur when two context states are joined together, which also have a particular logical connective including AND, NOT, OR. In Figure 9, we illustrate a context aggregation, and the aggregation rule that is created.

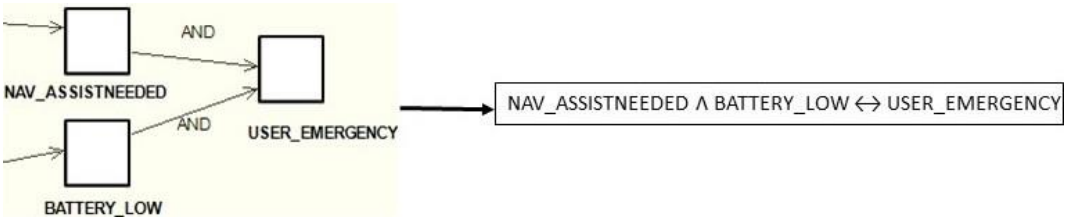



Figure 9 Aggregation Rule Generation (without temporal operators)

## 5. Context Reasoner

To support learning and reasoning across all POSEIDON compatible mobile applications, we propose a single context reasoning middleware, as shown in Figure 10. This middleware is opensource[1], and is distributed on the Google Play Store[2] for the Android platform.

[1] Source code can be found at: https://github.com/deankramer/POSEIDON-Context
[2] Found at: https://play.google.com/store/apps/details?id=org.poseidon_project.context

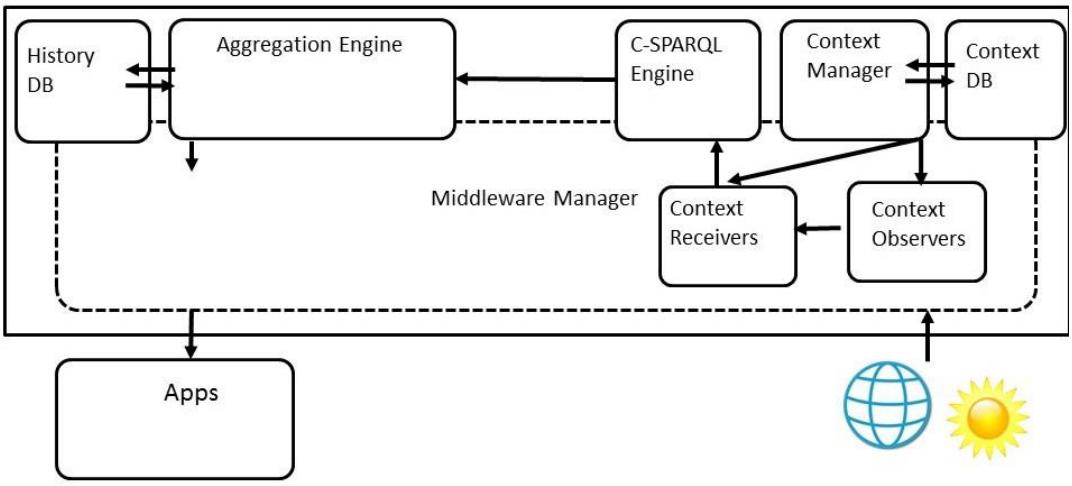



Figure 10 Context Middleware Architecture

The context reasoning middleware can be broken down into the following essential components:

- **ContextObservers:** These are the actual context acquisition components that collect the raw contextual data about the device, user, and environment. Different observers are introduced later in Section 4.1. Different applications can extend the middleware with new context observers.
- **ContextReceivers:** These are the components that receive the raw context data, and send structured RDF streams to the Context Reasoner. Different applications can extend the reasoner with new receivers to support newly included context observers.
- **Context Manager:** This component handles the runtime management of context observers and receivers based on the context requirements of the POSEIDON compatible applications. At runtime it can import, dynamically load, and remove context observers and receivers.
- **Context Reasoner:** This component handles the contextual reasoning and inference making. It relies on the C-SPARQL engine, to compliment and use the rules we define in C-SPARQL. Information on C-SPARQL rules is given in Deliverable 3.1 Ontology and Language.
- **Middleware Manager:** This act as a centralised component to handle communication between the reasoner and the POSEIDON compatible applications on the mobile device.

## 5.1 Context Manager

The context manager is responsible for handling the runtime management of all context observers and receivers.

### 5.1.1 Context Observer Database

To support runtime extensibility of context observers, the context manager has a database to contain different information required for runtime loading and instantiation. In the database, for every context observer, the following information is stored:

- **Id:** A unique identifier for the context component, stored as an integer.
- **Packagename:** This is the class namespace the context observer is part of, stored as a String. An example of this includes “org.poseidon_project.contexts.envir” as used in the Context Library.

- **Name:** This is the name of the context observer, used in the class definition of the observer, stored as a String. An example of this includes “BatteryContext”, and should match the name of the actual implementation class.
- **Owner:** The owner of the context observer is the application that deployed it, which is stored as a String.
- **Permission:** The permission of the context observer corresponds to whether the context component is private or public. By a context observer being public, it can be used by any application, compared with just the owner being able to use a private context observer. This is stored as an integer, where zero equates to a public context, and one equates to a private context observer.
- **Dex_file:** The dex file is the compiled unit that contains the context observers, and receiver. In Android, these classes are compiled to a .dex file, which are loaded at runtime.

### 5.1.2 Handling Context Observers

If a context observer is part of the reasoner, or deployed from a POSEIDON application, it can be used by the context manager. When the context reasoner requires raw context data to be collected, it creates a new context requirement. If there is an instance of the context observer already running, the identifier of the application requesting the requirement is then added to it. However, if the observer is not already running, it is dynamically loaded.

To dynamically load the context observer, the context manager gets the different information required from the context database, and instantiates it using Java reflection.

### 5.1.3 Handling Context Receivers

For new raw contextual data to be reasoned over, a new context receiver is required. This requires that when POSEIDON compatible applications use the context middleware, their context receiver (if one is imported) needs to be dynamically loaded at runtime. However, unlike the context observers that are completely self-contained, and run separately, to easy ensure all context data is processed by context receivers, we need to chain each receiver together. By chaining each receiver together, raw context data travels down the chain being processed by the appropriate receiver. To enable this functionality, we use the *Decorator* design pattern, which allows for behaviour to be added to individual class instances. Our use of this design pattern can be seen in Figure 11.

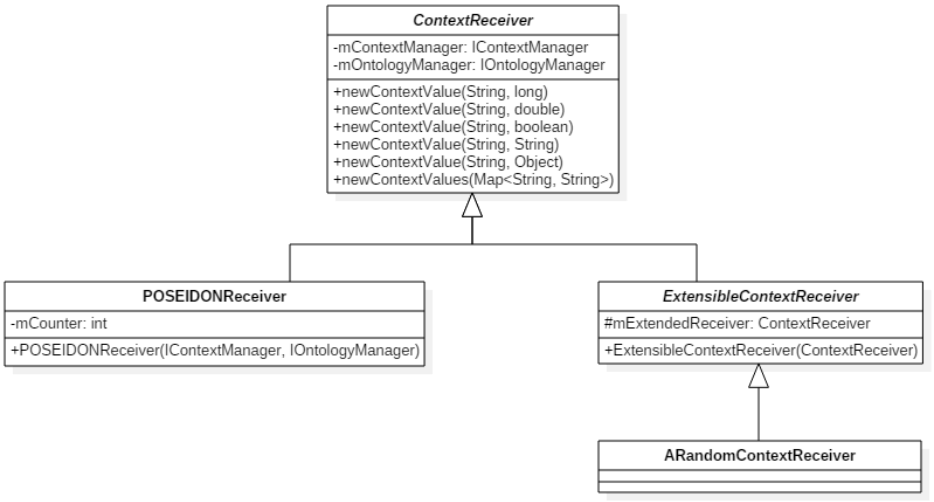



Figure 11 Decorator Design Pattern

At the top, we have an abstract *ContextReceiver* class that all classes inherit from. We have a concrete *POSEIDONReceiver* class, which includes all RDF stream operations required for built-in context observers. It is this class which forms the base class instance, which is decorated or wrapped. Next, we have an abstract *ExtensibleContextReceiver* decorator class, which is extended by other concrete classes provided by POSEIDON applications, noted in the diagram as *ARandomContextReceiver.*

When a new context receiver is loaded at runtime, we first get the runtime reference from the context manager for the current highest class instance in the chain. This could be just a reference to the POSEIDONReceiver, or a receiver from a different application. Next, we dynamically load and instantiate the new context receiver, passing this reference. This reference is added to the `mExtendedReceiver` class parameter, which is called after each method call. Once the class is instantiated and running, the context manager reference is updated to this new class instance, making new method invocations start at the top of the chain.

## 5.2 External Context Data

It is not always possible for the reasoning middleware to acquire all raw context data itself. This is because some contextual data can be based on state from different private applications. Therefore, the middleware can include context data sent from other apps in the mobile device. Context values being sent from external apps need to be handled in that application's ContextReceiver. This means, as the case with context observers, an application needs to have registered its ContextReceiver for context values to be reasoned over.

To send context values from external applications, this must be carried out using Android Intent Broadcasts. For the context receiver to receive the context, the intent must use the name `org.poseidon_project.context.EXTERNAL_CONTEXT_UPDATE`. These broadcasts are received by the context middleware, and then passed down the chain of context receivers.

## 5.3 C-SPARQL Engine

The context reasoner relies on the C-SPARQL engine (Barbieri *et al.*, 2010) for atomic context inferences. This engine is an extension of SPARQL thus requiring Apache Jena[3] libraries. To be compatible with Android, a number of different package refactoring was required. This is due to Android not allowing the use of javax.* packages as they count as internal APIs. These internal API are considered unsafe due to the higher possibility of changes between Android versions. Compounded with older versions of the libraries contained in the Android framework, namespace clashes and duplicate classes can occur. Also, due to Apache Jena being primarily designed for the Java Virtual Machine and not Android, we needed a few additional libraries to handle logging.

### 5.3.1 Registering Rules

In the context reasoner, to enable the use of C-SPARQL rules, the context reasoner has a number of methods to support the use of new rules. Each rule for example Listing 1, is part of a set of rules that correspond to a type of context. For example, we have different rules that are all related to weather conditions. When an application wants to know the weather context information for a particular location, the application just needs to ask for the context “weather” with the location it is interested in. For the first pilot, currently, all context rules are hardcoded in a class to register the rules, and start the appropriate context observers.

[3] http://jena.apache.org/

```
REGISTER QUERY weatherContextIsRaining AS
PREFIX ex: <http://ie.cs.mdx.ac.uk/POSEIDON/envir#>
CONSTRUCT { ex:weather <http://ie.cs.mdx.ac.uk/POSEIDON/context/is>
"WEATHER_RAINING"}
FROM STREAM <http://poseidon-project.org/context-stream>[RANGE 10s STEP 4s]
WHERE { ?m ex:hasTemperatureValue ?tempValueIRI .
        ?m ex:hasPrecipitationValue ?precipValueIRI .
        ?precipValueIRI ex:precipitationValue ?precipValue .
        ?tempValueIRI ex:temperatureValue ?tempValue .
      FILTER (?precipValue >= 0.1)
      FILTER (?tempValue >= 15)
      }
```

Listing 1 Raining C-SPARQL Rule

In the future, we plan to enable these rules to be added to the reasoner via JSON/XML files, which can be parsed and loaded at runtime.

### 5.3.2 Dealing with Context Changes

Each registered rule will execute based on its window/stepping specifications. If a rule does turn out to be true, the context string is sent, e.g. WEATHER_RAINING as in Listing 1. This triple including the context string is gathered by the ContextRuleObserver, which collects RDFs returned by C-SPARQL. The value of the string is then sent to the main context reasoner service as a context update.

In the context reasoner service, there is a table of context values. When a new context value is sent to the reasoner, it checks this table of values. If the value is a new one (either a context has no value, or the value is different), the table is updated, and the context value is broadcasted to applications.

## 5.4 Aggregation Engine

Inside the reasoner, the C-SPARL Engine is designed to handle atomic contexts, using data from various sources including phone sensors, Bluetooth devices, cloud services etc. This can provide some low level knowledge on the user and device, but by aggregating these conditions, we can form higher level context information. Aggregated rules supported are those proposed by Alegre *et al.* (2014). These aggregations are essentially propositional formulas, with the addition of temporal operators that can be applied to the different primitive variables in each rule.

### 5.4.1 Temporal Operators

The use of temporal operator is used with the rules to specify time related constraints on the contexts you wish to reason over. For example, you may wish to only consider if it was raining between 07:00 and 12:00. For each temporal operator we can apply stronger and weaker versions whereby ‘stronger’ we mean an assertion that is always true for the complete period of time, and by ‘weaker’ an assertion that can be true anytime within the stated period of time.

Temporal operators can be defined in an absolute manner (between a specific time and another time) and relative (a number of seconds in the past).

### 5.4.2 Execution

The execution of aggregation rules differs to atomic rules. Each time a context within the global reasoner service table changes, our system considers if any aggregation rules need to be fired. This is carried out by which when each aggregation rule is registered/active, we store the different antecedents that are used within that rules. By doing this, we only need to fire the rule if it is

affected by the context which has changed. This avoids every single aggregate rule firing each time a context value is changed in the system.

If a rule is found to require firing, we first need to update the antecedents in the rule. By doing this, we need to apply a Boolean value to each of them regarding if they are in fact true. For antecedents that do not contain temporal operators we check the global context table to see if their value is the most current. If it is not, or if the value of the context is unknown, we consider it false. For antecedents that do contain temporal operators, we will firstly check the context table to see if the timestamp of the context change answers the operator. If it does not, we then check the context history database and give it a true or false value. After all antecedents values are updated, we check to see if the rule is satisfiable using a SAT solver. We then check to see if the new value of the context is a change from the global context value table. Once all rules that need checking have done so, we see if any of the context values have changed, if there was a change, the process restarts taking into consideration the new global context values.

To improve the performance of the reasoning, when aggregate rules including temporal operators are registered, if we can evaluate antecedents already (perhaps we are checking absolute values in the past) we do so, and cache the result. This means that when the rule is evaluated, we do not need to check the context history database.

To evaluate the aggregation rules, we use the Prop4J[4] library, which uses Sat4J to handle propositional formulas.

## 5.5 Context Library – aContextLib

Before the development of this centralised system, all context observers were developed as part of the POSEIDON Context Library. To greater support modularity in different applications, we continue to develop the library, now named aContextLib [5], to better support developers in using different raw context data in their applications. Our centralised system also contains this library thus giving it ability to use the raw data it can collect with the inference rules. In Figure 13, we include a current diagram of aContextLib using the Unified Modelling Language (UML). Within the library there are two main types of components:

- **ContextObserver:** This abstract class contains method definitions required for each context acquisition component. The methods contained in this class are predominantly designed to handle the lifecycle of the context component.
- **ContextReceiver:** This is a collection of both an interface *IContextReceiver* and *ContextReceiver***.** As our library is built to support two functions: 1) allow developers to develop raw context data retrieval for the reasoner and 2) allow the developer more easily use the data in their own applications, even within the use of a reasoner, the two types are used depending on their particular use case. If the developer is developing for the reasoner, they much create a class that extends the *ContextReceiver* class with implementations that take the raw data, and inserts into to RDF for the reasoner. If the developer on the other hand is developing just a standalone application without the need for our reasoner, they just need to implement the *IContextReceiver* to receive the raw data callbacks.

[4] Available at: http://spl2go.cs.ovgu.de/projects/1
[5] Source code can be found at: https://github.com/deankramer/aContextLib

The base ContextObserver abstract class is such that it must be extended across all context acquiring components. Using this interface, we defined a set of different abstract context classes that can be extended by different concrete context components. These abstract context classes include:

- **BroadcastContext:** This class is designed to listen for context data changes from the devices which are broadcasted using Android Intent broadcasts. This context functions by implementing an Intent listener. An Intent listener then receives each Intent broadcast that fits a particular filter, for example `Intent.Action_BATTERY_CHANGED` for battery related updates. A concrete context extending this class is required for each type of Intent filter based context.
- **SensorContext:** This class collects context data from the device sensors. Each instance of this context calls the Android SensorManager to listen for specific sensors which are defined using Android Sensor class constants. By default, we have used the interval time of `SensorManager.SENSOR_DELAY_NORMAL`. This can be altered for individual contexts. A concrete sensor based context extending this class can only handle a single sensor at once.
- **PullObserver:** This context can be used for carrying out a specific task at defined time intervals. For example, if contexts are gathered from remote locations manually, this can be carried out using this context. This context by default collects contexts every 2 seconds, and can be changed by each context instance on instantiation and afterwards.
- **PushObserver:** This abstract class is a general class for all push based observers including *BroadcastContext, SensorContext, LocationContext*, and *BluetoothLEDevice***.**
- **LocationContext:** This class is designed to deal with contexts that require the use of the device's location services. This includes the use of the GPS/GLONASS receiver, Wireless Fidelity (Wi-Fi), and telephony triangulation. By default, each of the contexts use the GPS receiver, gets location data either every 3 seconds, or every 10 metres.
- **BluetoothLEDevice:** This context is used for data from different Bluetooth low energy devices, including heart rate monitors. This context collects raw bytes data from the Bluetooth Gatt, where in each concrete observer you fetch the contextual data.

As described above, these classes are abstract, and therefore must be extended by concrete context classes. Based on these abstract context classes, we currently have the following concrete context components:

- **BatteryContext:** Monitors the remaining amount of device battery in percent.
- **WifiContext:** Monitors the status of the WiFi receiver on the device, whether it is not active, disconnected, connecting, or connected to an access point.
- **TelephonyContext:** Monitors multiple parameters of the device's telephony receiver. This currently includes connection status, and whether the device is currently roaming.
- **CompassContext:** Monitors the current orientation of the device in degrees.
- **LightContext:** Monitors the current amount of ambient light, measured in lumens.
- **ExternalStorageSpaceContext:** Monitors the amount of storage space remaining on the devices shared storage area including the SD card, measured in megabytes (MB).
- **GPSIndoorOutdoorContext:** Monitors and deduces if the device is indoors or outdoors using the GPS receiver. This is carried out by sampling GPS signal/noise ratios.
- **WeatherContext:** Monitors and sends weather data for defined locations
- **BadWeatherContext:** Monitors and deduces whether the weather is expected to be cold and/or raining at the given locations.
- **StepCounter:** Monitors the current number of steps the user walks within a given interval.

- **DistanceTravelledContext:** Monitors the total distance the device moves within a given interval.
- **HeartRateMonitor:** Monitors the current heartrate a user wearing a Bluetooth heartrate monitor.

In addition to these observers, the library also has additional classes designed to assist the developer. These classes include:

- **ContextReasoner:** A helper class for end-user applications to connect to the context reasoner. Instead of the developer having to create their own connection, and handling file operations for importing new context observers and rules, this class handles this, while using simple Java methods and bindings.
- **InternetSource:** A helper class for developers to get specific data from internet sources. This is used for weather data for example. The developer can just implement the abstract methods within this class including any HTTP headers that might be required (for API keys for example).

Developers must include observer specific Android permissions before they are deployed, and runtime permissions are handled directly by each observer (if needed).

## 5.6 Context Settings

Using the context reasoner, different applications can get contextual information about the user, device, and environment. These context rules however are static in that particular values in the rules cannot be changed. In the areas of personalisation, to be better suit users' personal needs, we allow particular values within rules to be modified by the user in the R-Settings application, as shown in Figure 12. These settings are fully synchronised with the main POSEIDON cloud storage.

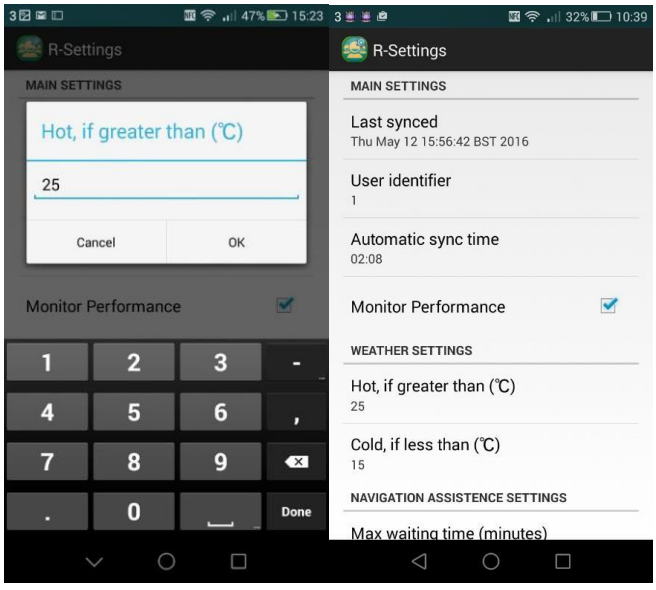



Figure 12 Context Personalisation

Currently, for rules already designed and included in the reasoner, we support the following personalisations:

- **Weather:** User can set their rated hot and cold temperatures.

- **Standstill:** User can set the maximum amount of time they can be relatively standstill. This is useful for when the user waits for the bus.
- **Deviations:** User can set the maximum number of small deviations in a route, before the user is offered assistance by the secondary user.

For developers to use personalisation in their rules, first the developers need to get the particular preference added to the central POSEIDON cloud (Tellu SmartPlatform). To do this, for each preference, they need to submit a key (how the preference is identified) and the value type (is it a number, String, Boolean etc).

Next, in the context models, the developer needs to alter their inference rules to include the preferences. This is done by altering static values in their inference rules to include the preference name with a prefix of "$$" to indicate it is a context preference. When the rule is registered it will use the value stored in this preference.

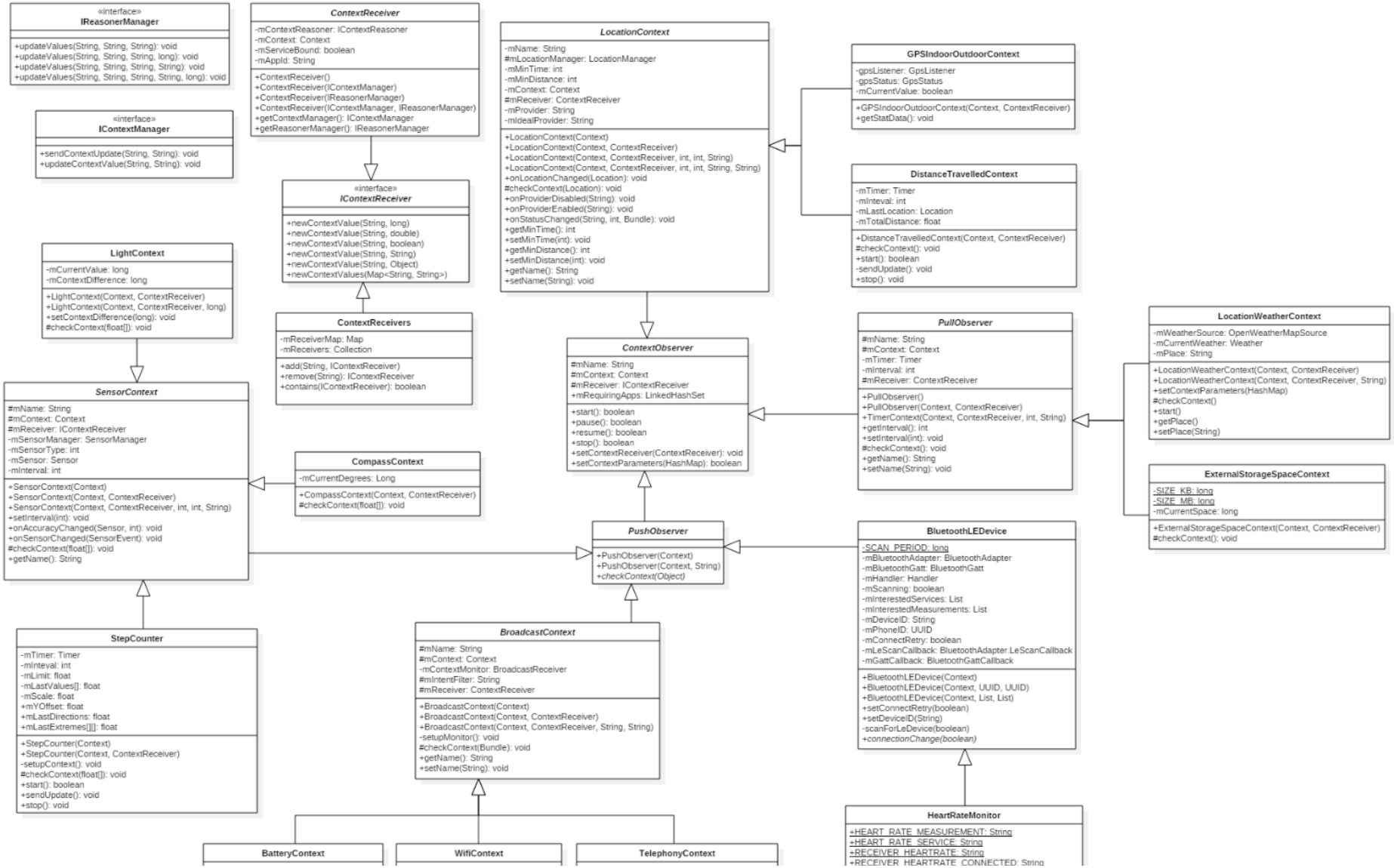



Figure 13 aContextLib UML Diagram

# 6. Learning Module

The context middleware already presented helps infer different contextual situations in both proactive and reactive situations. In addition to this, the system includes a learning module which can be used by both, primary and secondary users to learn and understand different parameters of the primary user over a sustained period of time.

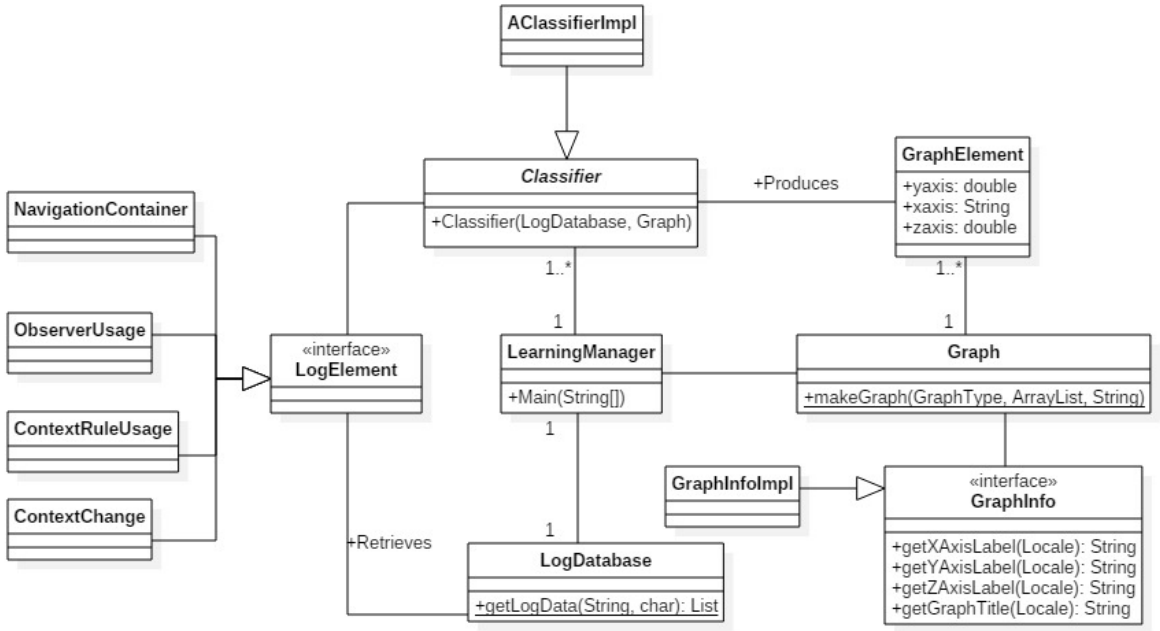



Figure 14 Learning Module

The learning module is made up of a collection of classes, as shown in Figure 14. The core classes required include:

- **LearningManager:** The main entry point to the service. It handles all calls, and returns a Base64 encoded graph as the result of the query based on the chosen query type
- **LogDatabase:** Handles raw data queries from the main database of different context activities. For the learning module to work, the user must have allowed the context data to be used by the service explicitly. This is set in the context preferences both on the mobile and the POSEIDON Carers web.
- **Graph:** Generates the specific visualisation of the query in question, currently in graphs and pie charts. This class requires specific graph data and the graph parameters e.g. labels. This information is held in classes implementing the GraphInfo interface.
- **Classifier:** This first gets the raw data from the context database based on the user and the time window selected, and then carries out its own data classification and sorting. Developers would need to create a class that extends this abstract class to enable new queries to be handled by the module.

Currently, our learning module supports the following types of queries:

- **Number of deviations and standstills:** Over the course of a primary user navigation, it is possible they may make small deviations from the planned route. Also, if they are waiting for public transport, it is possible that the user waits far too long for their connection service. We

class these different events as undesirable, and therefore track the number of times these happen to a user. This query can be used as a way of gauging how and if the primary user needs additional support with learning the route, or if a route requires alterations due to a persistent issue.

- **Journey Time:** We intend our navigation system being useful for the primary user to navigate to different events including school, work, and other leisure activities. Many of these activities often require the user to be at the location before a particular time (in the case of school and work). It is therefore desirable to be able to track how fast they do actually get to their destination. This can be a tool for finding if the user should leave earlier for a particularly regular journey.
- **User weight:** This shows the primary users weight over the different time scales, which can be used by the secondary user to know if dietary changes are required to support healthier living.

When querying the learning module web service, the following parameters are used:

- **API Key:** The specific API key identifying the service/application that wants to use the learning module service.
- **User**: The identifier for the user you wish to query about. The identifier is their unique POSEIDON username registered with the main systems.
- **Query**: The specific query identifier, which includes the specific classification and sorting logic.
- **Window**: The time window of data you wish to include in the query. Perhaps you wish to see results from data from the last day, week, or from the last month etc.
- **Other**: Other parameters strictly required by the query. For example for our rule that looks at the number of deviations and standstill contexts, you can query against particular routes used.
- **Language**: The language in which the visualisations should use, particularly for axis annotations in graphs.

## 6.1 Use by End-Users

For the secondary and primary users to use the learning module, they have to access through the POSEIDON Carer's Web[6]. Once the user has logged into the Carer's Web, they need to go to the **Settings** tab, along the top of the site.

The Learning module is in the lower half of the screen, and may require scrolling to reach, depending on the device as showing in Figure 15.

[6] https://ri.smarttracker.no/poseidon/

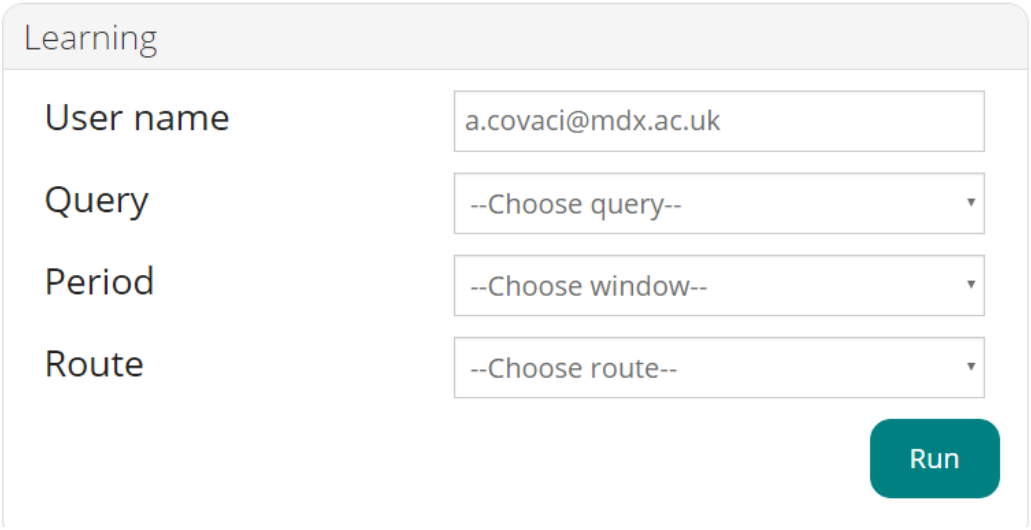
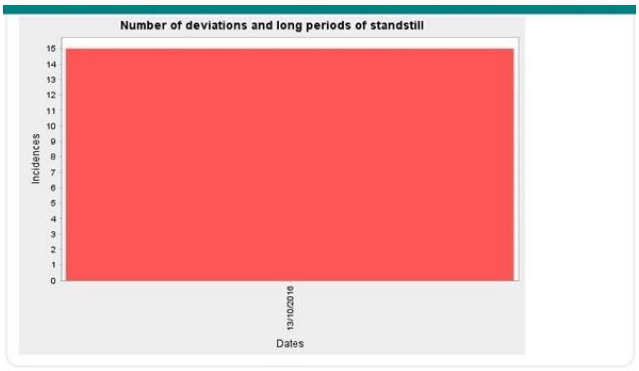


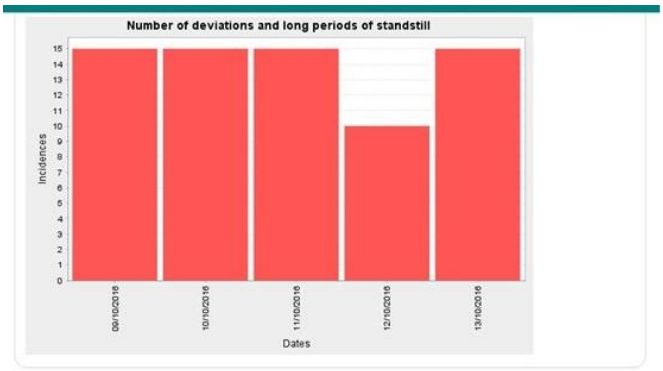

Figure 15 Learning Module Selection

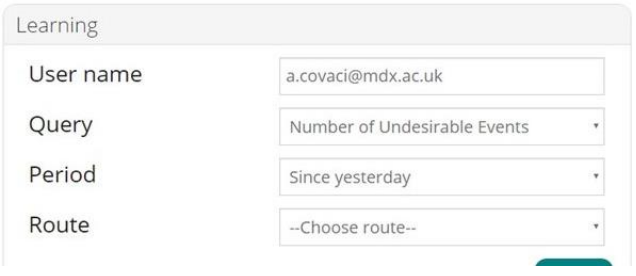
From here, the user can select the different parameter they are interested in include:

- Query: Which learning query do they want to run, picking one of those described earlier in this section of the deliverable.
- Period: The period of time they which to run the query against. This can include the last day, week, or month.
- Route: This allows the user to specify a particular route. In the case of route time, it makes sense to select the particular types of journey, or the graph will be a mix of all of the different journey times. If there is more than a single journey in a specific day, it will calculate the average time.

Below, we show the different periods for small deviations and standstills. These results could suggest that there has been an issue created alone the route. This could be caused roadworks and diversions, which would mean that the route designed by the secondary user needs adjustment.



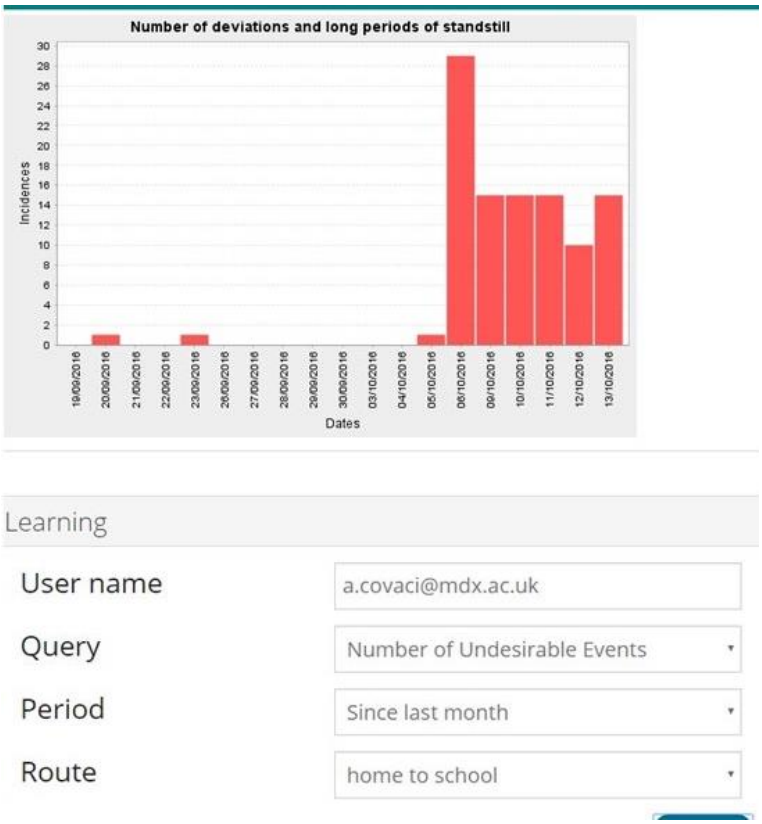
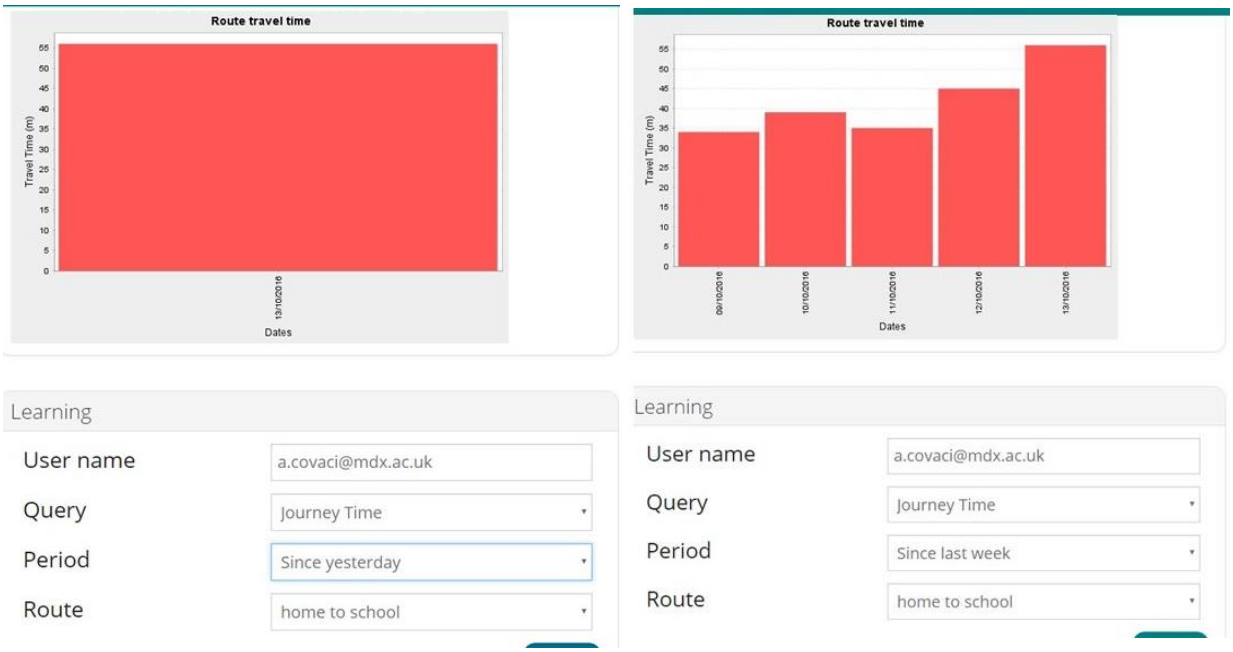




Figure 16 Month View of Deviations and Standstills

In addition, below we have screenshots of queries of journey time. As you can see, it would appear the user has a pattern of longer journey times through the week. The highest journey time could for example be attributed to it being Friday 13th, which some people with superstitions can act more cautiously.



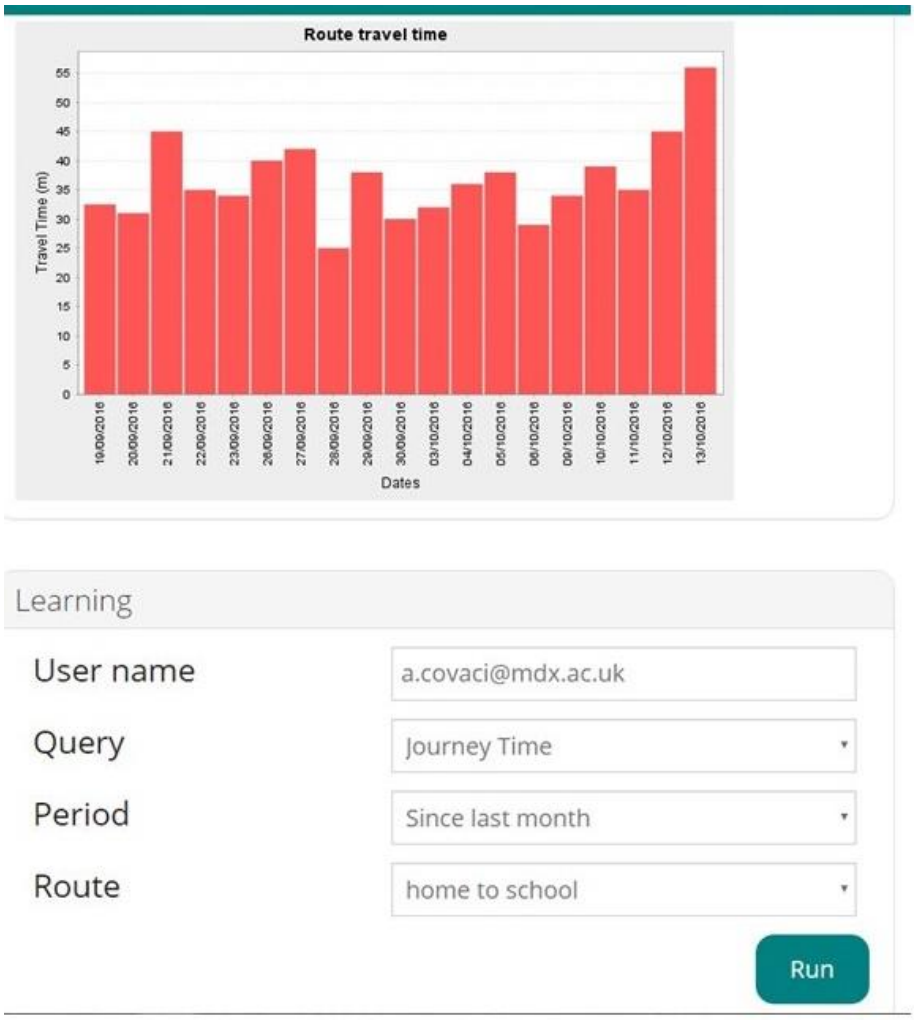



Figure 17 Monthly View of Travel Times

# 7. Conclusions

In this deliverable, we explored the state of the art in relation to the area of learning and reasoning for the POSEIDON project. Following this, the current vision for learning and reasoning within the project was described and discussed. This module can play a very large role within the primary users applications for both assisting them, and in the subject of privacy, assist the secondary user. We have propose a reasoning modelling notation which can be used for modelling the different context states and inference rules, which then auto generated the relevant C-SPARQL queries, as shown in D3.1. A context reasoning middleware, which has been used all POSEIDON pilots has been described. It uses both C-SPARQL for atomic level context inferences, and a temporal reasoning language for context aggregations. It also supports personalisation through the personalisation of values within context rules. Finally, we described a new learning module designed to help the primary and secondary user learning about different statistics of the primary user. These can be used to either indicate more practicing is required, or alterations are required for a given route.

# References

Alegre, U., Augusto, J. C. and Aztiria, A. (2014). Temporal Reasoning for Intuitive Specification of Context-Awareness. In: *Proceedings of the 2014 International Conference on Intelligent Environments (IE)*, 2014.

Allen, J. F. and Ferguson, G. (1994). Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4, p.531–579. [Online]. Available at: doi:10.1093/logcom/4.5.531.

Augusto, J. C. (2007). Ambient Intelligence: The Confluence of Ubiquitous/Pervasive Computing and Artificial Intelligence. In: *Intelligent Computing Everywhere*, p.213–234. [Online]. Available at: doi:10.1007/978-1-84628-943-9_11.

Augusto, J. C., Callaghan, V., Cook, D., Kameas, A. and Satoh, I. (2013). Intelligent Environments: a manifesto. *Human-centric Computing and Information Sciences*, 3, p.12. [Online]. Available at: doi:10.1186/2192-1962-3-12.

Augusto, J. C., Huch, M., Kameas, A., Maitland, Julie and McCullagh, P., Roberts, J., Sixsmith, A. and Wichert, R. (2012). *Handbook on Ambient Assisted Living*. IOS Press.

Augusto, J. C., Liu, J., McCullagh, P., Wang, H. and Yang, J.-B. (2008). Management of Uncertainty and Spatio-Temporal Aspects for Monitoring and Diagnosis in a Smart Home. *International Journal of Computational Intelligence Systems*, 1 (4), p.361–378. [Online]. Available at: doi:10.2991/ijcis.2008.1.4.8.

Aztiria, A., Augusto, J. C., Basagoiti, R., Izaguirre, A. and Cook, D. J. (2013). Learning Frequent Behaviors of the Users in Intelligent Environments. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43 (6), p.1265–1278.

Aztiria, A., Izaguirre, A. and Augusto, J. C. (2010). Learning patterns in ambient intelligence environments: A survey. *Artificial Intelligence Review*, 34, p.35–51. [Online]. Available at: doi:10.1007/s10462-010-9160-3.

Barbieri, D. F., Braga, D., Ceri, S., Valle, E. Della and Grossniklaus, M. (2010). C-SPARQL: A Continuous Query Language for RDF Data Streams. *International Journal of Semantic Computing*, 4 (1), World Scientific Publishing Company, p.3–25. [Online]. Available at: doi:10.1142/S1793351X10000936 [Accessed: 20 March 2015].

Benerecetti, M., Bouquet, P. and Ghidini, C. (2000). Contextual reasoning distilled. *Journal of Experimental & Theoretical Artificial Intelligence*, 12 (3), p.279–305. [Online]. Available at: doi:10.1080/09528130050111446.

Cafezeiro, I., Viterbo, J. and Rademaker, Alexandre Haeusler Edward Hermann Endler, M. (2008). A Formal Framework for Modeling Context-Aware Behavior in Ubiquitous Computing. In: *Leveraging Applications of Formal Methods, Verification and Validation*, 17, p.519–533.

Cardelli, L. (1999). Mobility and Security. In: *Lecture Notes for the Marktoberdorf Summer School 1999*.

Chen, H., Perich, F., Finin, T. and Joshi, A. (2004). SOUPA: standard ontology for ubiquitous and pervasive applications. In: *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, August 2004, p.258–267. [Online]. Available at: doi:10.1109/MOBIQ.2004.1331732.

Daniele, L. M. (2006). *Towards a Rule-based Approach for Context-Aware Applications*. University of Cagliari.

Daniele, L. M., Silva, E., Pires, L. F. and Van Sinderen, M. (2009). A SOA-based platform-specific framework for context-aware mobile applications. In: *Lecture Notes in Business Information Processing*, 38 LNBIP, 2009, p.25–37. [Online]. Available at: doi:10.1007/978-3-642-04750-3_3.

Dargie, W. (2007). The role of probabilistic schemes in multisensor context-awareness. In: *Proceedings of the 5th Annual IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2007*, 2007, p.27–32. [Online]. Available at: doi:10.1109/PERCOMW.2007.115.

Dey, A. K. (2001). Understanding and Using Context. *Personal and Ubiquitous Computing*, 5, p.4–7. [Online]. Available at: doi:10.1007/s007790170019.

di Flora, C. and Hermersdorf, M. (2008). A practical implementation of indoor location-based services using simple WiFi positioning. *Journal of Location Based Services*, 2, p.87–111. [Online]. Available at: doi:10.1080/17489720802415205.

Fonteles, A. S., Neto, B. J. ., Maia, M., Viana, W. and Andrade, R. M. . (2013). An Adaptive Context Acquisition Framework to Support Mobile Spatical and Context-Aware Applications. *Web and Wireless Geographical Information Systems*, p.100–116.

Golab, L. and Ozsu, M. T. (2003). Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams. In: *VLDB*, (February), 2003, p.500–511. [Online]. Available at: http://portal.acm.org/citation.cfm?id=1315451.1315495.

Gueffaz, M., Rampacek, S. and Nicolle, C. (2012). Temporal logic to query semantic graphs using the model checking method. *Journal of Software*, 7 (7), p.1462–1472. [Online]. Available at: doi:10.4304/jsw.7.7.1462-1472.

Heckmann, D., Schwartz, T., Brandherm, B., Schmitz, M. and von Wilamowitz-Moellendorff, M. (2005). GUMO: The General User Model Ontology. In: *10th International Conference on user Modeling*, 2005, p.428–432.

Held, A., Buchholz, S. and Schill, A. (2002). Modeling of Context Information for Pervasive Computing Applications. In: *Proc. of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI2002)*, 2002, p.6 pp. [Online]. Available at: http://citeseer.ist.psu.edu/held02modeling.html.

Henricksen, K. and Indulska, J. (2004). Modelling and using imperfect context information. In: *IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second*, 2004. [Online]. Available at: doi:10.1109/PERCOMW.2004.1276901.

Henricksen, K. and Indulska, J. (2006). Developing context-aware pervasive computing applications: Models and approach. *Pervasive Mob. Comput.*, 2 (1), Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V., p.37–64.

Henricksen, K., Indulska, J. and Rakotonirainy, A. (2002). Modeling Context Information in Pervasive Computing Systems. In: *Proceedings of the First International Conference on Pervasive Computing*,

2002, p.167–180. [Online]. Available at: doi:10.1007/3-540-45866-2_14.

Hoyos, J. R., García-Molina, J. and Botía, J. A. (2013). A domain-specific language for context modeling in context-aware systems. *Journal of Systems and Software*, 86 (11), p.2890–2905. [Online]. Available at: doi:http://dx.doi.org/10.1016/j.jss.2013.07.008.

Katsiri, E., Seranno, J. M. and Serrat, J. (2010). Application of Logic Models for Pervasive Computing Environments and Context-Aware Services Support. In: *Multimedia Services in Intelligent Environments*, 2, p.105–117.

Kikiras, P., Tsetsos, V. and Hadjiefthymiades, S. (2006). Ontology-Based User Modeling for Pedestrian Navigation Systems. In: *Proceedings of the ECAI 2006 Worshop on Ubiquitous User Modeling (UbiqUM 2006)*, 2006, Riva del Garda, Italy, p.1–6.

Kjærgaard, M. B. and Bunde-pedersen, J. (2006). Towards a Formal Model of Context Awareness. In: *Proceedings of the International Workshop of Combining Theory and Systems Building in Pervasive Computing-Pervasive 2006*, 2006, p.667–674.

Kramer, D., Augusto, J. C. and Clark, T. (2014). Context-Awareness to Increase Inclusion of People with DS in Society. *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*. [Online]. Available at: http://www.aaai.org/ocs/index.php/WS/AAAIW14/paper/view/8715 [Accessed: 15 April 2015].

Kramer, D., Kocurova, A., Oussena, S., Clark, T. and Komisarczuk, P. (2011). An extensible, self contained, layered approach to context acquisition. In: *Proceedings of the Third International Workshop on Middleware for Pervasive Mobile and Embedded Computing - M-MPAC '11*, 12 December 2011, New York, New York, USA: ACM Press, p.1–7. [Online]. Available at: doi:10.1145/2090316.2090322 [Accessed: 9 April 2014].

Librelotto, G., Augustin, I., Gassen, J., Kurtz, G., Freitas, L., Martini, R. and Azevedo, R. (2010). OntoHealth: An Ontology Applied to Pervasive Hospital Environments. *International Journal of Advanced Pervasive and Ubiquitous Computing*, 2 (3), IGI Global, p.33–45. [Online]. Available at: doi:10.4018/japuc.2010070103 [Accessed: 3 April 2014].

Lim, B. and Dey, A. (2010). Toolkit to support intelligibility in context-aware applications. In: *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, 2010, p.13–22. [Online]. Available at: doi:10.1145/1864349.1864353.

Meditskos, G., Dasiopoulou, S., Efstathiou, V. and Kompatsiaris, I. (2013). SP-ACT: A hybrid framework for complex activity recognition combining OWL and SPARQL rules. In: *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, March 2013, IEEE, p.25–30. [Online]. Available at: doi:10.1109/PerComW.2013.6529451 [Accessed: 23 September 2014].

Muhlenbrock, M., Brdiczka, O., Snowdon, D. and Meunier, J.-L. (2004). Learning to detect user activity and availability from a variety of sensor data. In: *Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. Proceedings of the*, 2004. [Online]. Available at: doi:10.1109/PERCOM.2004.1276841.

Muñoz, J., Pelechano, V. and Fons, J. (2004). Model Driven Development of Pervasive Systems. In: *Proceedings of the 1st International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES)*, 2004, p.2–14.

Preuveneers, D. and Berbers, Y. (2005). Automated Context-Driven Composition of Pervasive Services to Alleviate Non-Functional Concerns. *International Journal of Computing & Information Sciences*, 3 (2), p.19–28.

Ramos, C., Augusto, J. C. and Shapiro, D. (2008). Ambient Intelligence: The Next Step for Artificial Intelligence. *IEEE Intelligent Systems*, 23, p.15–18. [Online]. Available at: doi:10.1109/MIS.2008.19.

Ranganathan, A., Mcgrath, R. E., Campbell, R. H. and Mickunas, M. D. (2003). Ontologies in a Pervasive Computing Environment. In: *Workshop on ontologies and distributed systems (part of the 18th International Join Conference of artificial Intelligence*, 2003.

Reichle, R., Wagner, M., Khan, M. U., Geihs, K., Lorenzo, J., Valla, M., Fra, C., Paspallis, N. and Papadopoulos, G. A. (2008). A Comprehensive Context Modeling Framework for Pervasive Computing Systems. In: *Distributed Applications and Interoperable Systems*, Lecture Notes in Computer Science, 5053, Springer Berlin Heidelberg, p.281–295. [Online]. Available at: doi:10.1007/978-3-540-68642-2_23.

Reignier, P., Brdiczka, O., Vaufreydaz, D., Crowley, J. and Maisonnasse, J. (2007). Context-Aware Environments: From Specification to Implementation. *Expert Systems*, 24 (5), p.305–320.

Reiter, R. (2001). On knowledge-based programming with sensing in the situation calculus. *ACM Transactions on Computational Logic*, 2 (4), ACM, p.433–457. [Online]. Available at: doi:10.1145/383779.383780 [Accessed: 24 September 2014].

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach (Second Edition)*. [Online]. Available at: doi:10.1016/0004-3702(96)00007-0.

Sadri, F. (2011). Ambient Intelligence: A Survey. *ACM Computing Surveys*, 43, p.1–66. [Online]. Available at: doi:10.1145/1978802.1978815.

Santofimia, M. J., Moya, F., Villanueva, F. J., Villa, D. and Lopez, J. C. (2009). An agent-based approach towards automatic service composition in ambient intelligence. *Artificial Intelligence Review*, 29 (3–4), p.265–276. [Online]. Available at: doi:10.1007/s10462-009-9145-2 [Accessed: 3 April 2014].

Serral, E., Valderas, P., Muñoz, J. and Pelechano, V. (2008). Towards a Model Driven Development of Context-aware Systems for AmI Environments. In: *Developing Ambient Intelligence*, Springer Paris, p.113–124. [Online]. Available at: doi:10.1007/978-2-287-78544-3_11.

Sheng, Q. Z. and Benatallah, B. (2005). ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services Development. In: *Proceedings of the International Conference on Mobile Business*, 2005, Washington, DC, USA: IEEE Computer Society, p.206–212.

Siewe, F., Zedan, H. and Cau, A. (2011). The Calculus of Context-aware Ambients. *Journal of Computer and System Sciences*, 77 (4), p.597–620. [Online]. Available at: doi:10.1016/j.jcss.2010.02.003 [Accessed: 31 March 2014].

Stavropoulos, T. G., Vrakas, D., Vlachava, D. and Bassiliades, N. (2012). BOnSAI : a Smart Building Ontology for Ambient Intelligence. In: *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, 2012.

Strang, T. and Linnhoff-Popien, C. (2004). A Context Modeling Survey. In: *Proceedings of the Workshop on Advanced Context Modelling, Reasoning and Management*, Workshop o, 2004, p.1–8. [Online]. Available at: doi:10.1.1.2.2060.

Wang, X. H., Zhang, D. Q., Gu, T. and Pung, H. K. (2004). Ontology Based Context Modeling and Reasoning Using OWL. In: *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, PERCOMW '04, 2004, Washington, DC, USA: IEEE Computer Society, p.18--. [Online]. Available at: http://dl.acm.org/citation.cfm?id=977405.978618.

Zografistou, D. (2012). *Support for context-aware healthcare in Ambient Assisted Living*. University of Crete.