Scientific Research Publishing

Journal of Computer and Communications, 2017, 5, 29-43
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Model of a Context-Aware Middleware for Mobile Workers

Esraa Moustafa, Gaëtan Rey, Stéphane Lavirotte, Jean-Yves Tigli

Université Côte d'Azur, Centre National de la Recherche Scientifique (CNRS), I3S, Nice, France
Email: esraa.maher@gmail.com, Gaetan.Rey@unice.fr, Stephane.Lavirotte@unice.fr, Jean-Yves.Tigli@unice.fr

How to cite this paper: Moustafa, E., Rey, G., Lavirotte, S. and Tigli, J.-Y. (2017) Model of a Context-Aware Middleware for Mobile Workers. *Journal of Computer and Communications*, **5**, 29-43.
https://doi.org/10.4236/jcc.2017.54003

Received: January 26, 2017
Accepted: March 11, 2017
Published: March 14, 2017

Abstract

With the development of Internet of things and Web of things, computing becomes more pervasive, invisible and present everywhere. In fact, in our environment, we are surrounded by multiple devices that deliver (web) services which meet the needs of the users. However, the mobility of these devices as the users has important repercussions that challenge software design of these applications because the variability of the environment cannot be anticipated at the design time. Thus, it will be interesting to dynamically discover the environment and adapt the application during its execution to the new contextual conditions. We therefore, propose a model of a context-aware middleware that can address this issue through a monitoring service which is capable of reasoning and observation channels capable of calculating the context during the runtime. The monitoring service evaluates the pre-defined X-Query predicates in the context manager and uses Prolog to deduce the services needed to respond back. An independent observation channel for each different predicate is then dynamically generated by the monitoring service depending on the current state of the environment. Each channel sends its result directly to the context manager which consequently calculates the context based on all the predicates' results while preserving the reactivity of the self-adaptive system.

Keywords

Auto-Adaptation, Context-Awareness, Middleware, Reasoning Engine

1. Introduction

Nowadays, the world has nearly 13 billion connected devices in use and it's estimated that by 2020, this number will increase to reach more than 29 billion connected devices [1]. Pervasive Computing, aka Ubiquitous Computing, was once a vision of Mark Weiser. He predicted that technologies would weave them-

selves into the fabric of everyday life until they were indistinguishable from it [2].

Despite the huge number of connected devices, pervasive computing is challenged by the constant change in the environment. People (along with their on-body sensors) are constantly moving; objects might get displaced and sensors might fail. These changes are impossible to predict at applications' design time. Thus, we need to discover the environment (context) dynamically and adapt the applications in the runtime accordingly through a Context-Aware Middleware.

This paper is organized as follows. Section 2 provides a discussion about related works. In Section 3, we introduce CONTINUUM project and the motivation behind our research. In Section 4, we discuss our contribution to the improvement of CONTINUUM. In Section 5, we introduce our model of a context-aware middleware from the conceptual and implementation perspectives. We also highlight our proof of concept. Finally, we conclude our work in Section 6.

2. Related Work

Many researchers proposed Context-Aware Middleware. In [3], Context-Awareness is defined as an "Opportunistic Situation Identification". A phone-centric middleware is proposed which uses a hybrid learning approach for reasoning that combines pre-computed models and imputed models. This middleware tackles multiple challenges specially the constant movement of the users but it's limited by the phone constraints such as the battery life and the local storage capacity. In [4], middleware architecture for a context-aware system in smart home environment is proposed. Web Ontology Language (WOL) is used to model the context and a rule-based algorithm is used for reasoning. A conducted survey on context aware middleware architectures [5] compared multiple recent projects such as FlexRFID [6], Context Awareness for Internet of Things (CA4IOT) [7] and Octopus [8]. Each project has a different modelling and reasoning approach. The survey pointed some limitations such as the need to handle the users' security, increase the degree of context awareness and ensure standardization. However, these middleware architectures are either based on predefined rules only and thus can't handle unknown situations or they use a specific language such as OWL which makes it hard for the users to define new situations without the engagement of developers.

3. CONTINUUM

CONTINUUM [9] Project offers architecture to evaluate the context in the runtime through a three-tiered adaptation middleware and observation channels that monitor the environment as shown in **Figure 1**. This architecture is based on Rasmussen's model of human cognitive behavior [10].

- Reflexive Behavioral Adaptation: The first tier can make the adaptation of the application according to the devices present in the environment and the rules of adaptations activated within a very short period of time hence the name "reflexive". This corresponds to human behaviors based on reflexes.
- Contextual Tactics Adaptation: The second tier identifies the current state

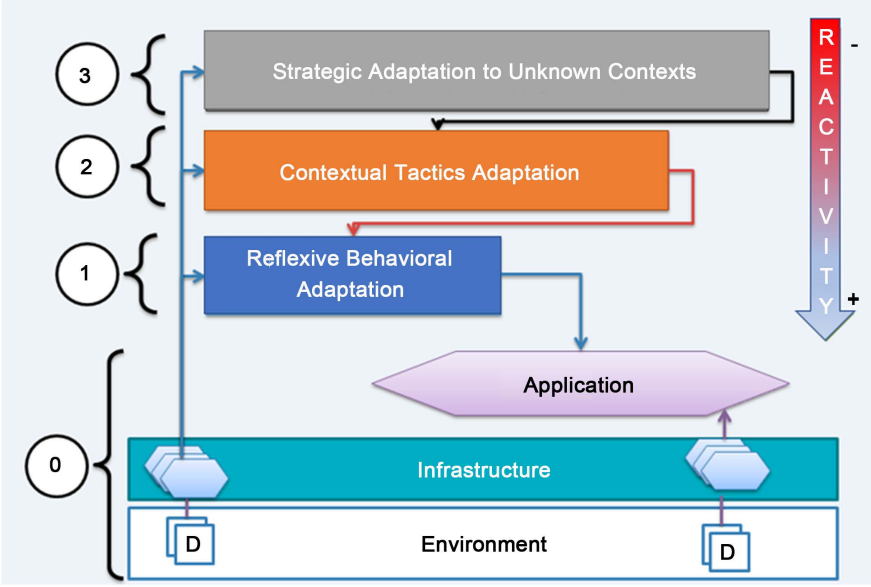

Figure 1. The architecture of CONTINUUM based on Rasmussen's model of cognitive behavior [4].

among a set of known states specified at the design through a knowledge base and a context manager. This corresponds to human behavior based on sets of rules learned with experience.

• Strategic Adaptation to Unknown Contexts: This layer proposes mechanisms to adapt the application to unknown states. This corresponds to human behavior based on knowledge. This involves developing an action plan to respond to an unknown situation.

3.1. Terminology

In the next sections, we'll be describing our work using below terms:

All the objects, people and computer systems are included in the environment.

The observables are the data of the devices (sensors) which make it possible to describe the attributes of the entities of the world (user, physical object ...). These observables serve to characterize the entities of the world (at the design time) and to identify them (at the runtime).

Predicates are Boolean functions based on entities. From a set of verified predicates, we find the current state of the context (usually named situation). It is for this reason that from n predicates, we will obtain 2^n possible states. For each state, we define one or more adaptation rules.

3.2. Scenario

To better illustrate the importance of a context-aware middleware for mobile workers, let's look at below scenario.

Raoul is a technician at the combustion turbine. Every morning, he makes a round to take measurements of some equipment in different sites. Each site has a different security level. There are 2 types of security:

• Building Security which indicates the level of authorization of the worker to access the site.

• Worker Security indicating if he is sufficiently equipped to access the site.

In this example, the goal is to identify the authorized and equipped worker to access any site. If this is the case, the time spent at the site is calculated, and an optimized itinerary service is offered to the technician, in order to increase the number of items he can inspect during his round. In case an issue occurred and the sound sensors, temperature or pressure exceeded a certain threshold, then an alert service notifies the technician to take the necessary measures. For simplicity purpose, we will limit ourselves to the pressure sensors in the scenario. In case there is an intruder, the system will launch an alarm.

In other words, with the notions defined in 2.1, we get the following:

For the security preoccupation context, we have the following predicates, states, and rules:

Predicates:

$P1$ is Zone Secured.

$P2$ is Authorized/Equipped Worker.

$P3$ Pressure Exceeded Threshold.

Rules:

$R1$ starts an alarm.

$R2$ calculates past time.

$R3$ shows route/itinerary service.

$R4$ shows Alert Service.

Since for each n predicates, we have 2^n possible states. We thus distinguish 8 possible states.

States 1 and 2

If Raoul enters either a secured or unsecured area, if he is authorized and equipped and the pressure sensor has exceeded the threshold, then the 3 rules ($R2$, $R3$ and $R4$) should be applied as in (1).

$$P1, P2, P3/!P1, P2, P3 \to R2, R3, R4. \tag{1}$$

States 3 and 4:

Same as states 1 and 2 except that the pressure doesn't exceed the threshold, $R2$ and $R3$ should thus be applied as in (2)

$$P1, P2, !P3/!P1, P2, !P3 \to R2, R3. \tag{2}$$

States 5 and 6:

If Raoul is unauthorized to enter a secured zone, an alarm should be started as in (3).

$$P1, !P2, P3/P1, !P2, !P3 \to R1. \tag{3}$$

States 7 and 8:

Same as states 5 and 6 except that Raoul is unauthorized to enter the unsecured zones as in (4):

$$!P1, !P2, !P3/!P1, !P2, P3 \to R1. \tag{4}$$

3.3. Motivation

Our research was driven by the results of CONTINUUM specifically by the 2nd tier. In fact, in this tier, the adaptation is performed in 3 steps [11] as shown in Figure 2.

1. Observation: The purpose of this step is to provide numerical observables. It relies on both digital and analogue sensors and converts their data into a format compatible with the next step.

2. Transformation: This step transforms numerical observables into semantic observables.

3. Identification: this stage makes it possible to identify the current situation from the semantic observables.

To make this happen, a Knowledge Base (KB) and a Context Manager (CM) were implemented [12].

The KB [13] is part of the Transformation step. It aims to maintain a model of the environment using ontology based RDFS language. It receives the values and events of the sensors as well as their appearance and disappearance and thus, it keeps an image of the entire system.

The CM is part of the identification step. It mainly interrogates the KB through queries to verify the different predicates.

Field experiments were performed with the help of mobile workers to evaluate CONTINUUM which showed very promising results [14]. On the other hand, some limitations were identified which can be summarized in below points:

1. The Knowledge Base (KB) is a centralized module and thus it's a single point of failure on which the system depends. In addition to that, it doesn't en sure

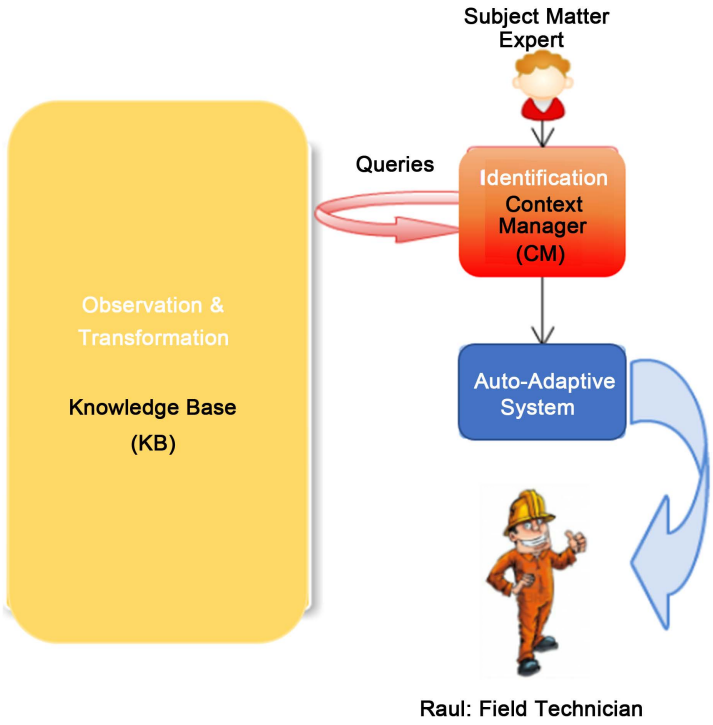

Figure 2. "Contextual tactics adaptation" Tier in CONTINUUM.

high availability nor scalability.

2. The Context Manager (CM) depends on the model of the KB. In CONITINUIUUM, the KB stores all the sensors' data using RDFS Ontology, consequently the predicates in CM must use the same ontology as the KB to be able to communicate with it. This limits the possibility of adding different reasoning engines.

4. Improving CONTINUUM

To start tackling these limitations, we divided our work into 3 parts:

1. Study of Predicates and its possible alternatives.
2. Study of the possible languages to use.
3. Study of the path between the observables and the predicates.

4.1. Study of Predicates and Its Possible Alternatives

In continuum, as explained above, situations are identified by evaluating the answer of the predicates for a given Preoccupation. But are these predicates sufficient to identify all possible situations for all the preoccupations?

To answer this question, we studied below possibilities:

i) Using continuous functions and operators to transform the output into Boolean.

For instance, a function can be used to transform the temperature data from the sensor into Fahrenheit degrees and then transform these values into Boolean values to decide if it's hot or not (Bayesian Networks can be used in this case).

ii) Using continuous functions and operators to transform the output into discrete values.

If we take the same temperature example, the only difference would be to use fuzzy logic for instance to transform the continuous values into discrete ones.

iii) Using predicates only.

Predicates offer a finite number of possibilities to identify the current situation since for n predicates we have 2^n states.

Also, it will be easier for an expert to express his rules associated to specific predicates.

Since our model is intended for experts not developers, we opted for the predicates approach.

4.2. Study of the Possible Languages to Use

Defining a language is essential to expressing the users' needs through predicates. This language must be independent of the reasoning techniques used.

In addition, we must take into consideration that the context is dynamic, that objects can be moved, people enter and leave rooms, temperatures fluctuate, and activities begin and end [15]. Also, we opt for a distributed approach to overcome continuum's limitations.

We found that Contextual Query Language (CQL) can be classified into 5 categories [16]:

SQL

The SQL works well with the relational model and therefore a SQL query only returns tabular data. However, the structure of a sensor network is hierarchical, so we need a query language that can traverse in a tree-like manner [17].

Ontology

We have found several context models based on OWL (Web Ontology Language) ontologies. OWL is much more expressive compared to other ontologies languages like RDFS and it allows semantic interoperability to exchange and share knowledge of the context between different systems in various domains [18]. On the other hand, their query language is specific and not generic.

Programming language

Programming languages can address various recursive queries, but they require the development of complex algorithms to handle queries [16]. We have eliminated it because it requires developers to write predicates and thus it will not be suitable for experts.

Graphic

We found a language called "chiromancer" [19] that is a Visual Query Language (VQL) based on a "Query by Browsing" paradigm. This paradigm is based on the questioning of a relational schema using familiar concepts of the desktop user interface paradigm. This is done by representing both the schema and the query as a folder hierarchy.

XML

Among the XML-based CQLs, the one that has most interested us is called "X-Query" because it is a flexible language by nature and it can query tabular and hierarchical data and is therefore adapted for the networks of Sensors. Moreover, it allows scalability and it meets our criteria of independence between the technology of the implementation of the reasoning and that of the writing of the requests. Moreover, it is easy for an expert to learn such a language.

4.3. Study of the Path between the Observables and the Predicates

This task consists in studying how predicates will be evaluated using the current observables in the environment. To do this, we studied the different types of reasoning to identify situations [20]. In fact, there are 2 approaches to reasoning.

The specification-based approach consists of reasoning according to predefined rules and predicates. For example, in the mobile worker scenario, if the pressure exceeded a certain threshold, an alert service will be activated.

This approach has the advantage that it is direct and easy to use by experts but its disadvantage is that it does not allow the identification of unknown situations and it is frustrating in cases where it is necessary to define many rules.

The learning-based approach consists in using learning techniques with the advantage of identifying unknown situations or better adapting to a given concern. For example, the user's mood is among the contexts extremely difficult to detect [21]. It is difficult to write rules to predict the mood of the user, since each user is different. That is why in this case, it is better to use learning techniques. These techniques learn to identify the situations from a training period in which

one or more people define situations according to the sensor data (Supervised Learning). There are also other learning algorithms that can do clustering but do not label them (Unsupervised Learning). Consequently, the major disadvantage is that learning can take a lot of human effort and time (several months [22]).

Since both approaches have advantages and disadvantages, the best is to use a hybrid approach [21].

5. Proposed Model

In the first part, we will present the conceptual model that we propose to answer the problem. Then, we will detail how to implement such a model. Finally, we present the proof of concept that we implemented.

5.1. Conceptual Model

Our model, as shown in **Figure 3**, is based on the model defined in continuum. It contains the context manager whose purpose, is to modify the list of adaptive rules (AR) activated in the self-adaptive system. The list of activated ARs is calculated per the current situation itself calculated according to the predicates.

To evaluate each of the predicates, the CM relies on observation channels. It will therefore start by looking for an observation channel corresponding to the predicate it's evaluating. If it does not find such a channel, it will ask its creation to the observation service.

Our model again introduces 2 essential components:

1. OBSERVATION SERVICE (OS) which receives the predicate from the Context Manager. This service is responsible for:

• Semantic alignment which is one of the scientific challenges of today. In fact, let us imagine that to express the first predicate of the scenario "is Zone Secured", we expressed it with another equivalent word, that is to say that we re

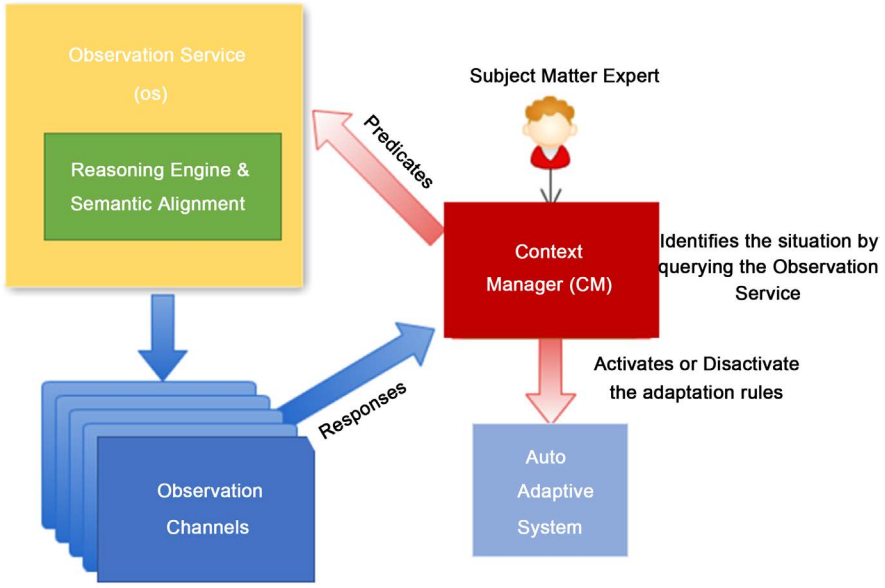

Figure 3. Conceptual model.

placed the word "zone" by "site" or "factory" or "place", the system will not know that it is the same predicate! This is why the system must take into consideration these equivalences; it must be able to discover correspondences between two different ontologies. This issue is currently attracting the attention of researchers in web semantic. We did not handle it as it was out of our scope.

• The reasoning needed to answer the predicate. In fact, this service contains a reasoning engine composed of 2 elements. The first is an observation base that keeps a model of the environment containing the description of existing business services or services from devices typically sensors or communicating objects. This database is updated automatically when a service appears or disappears. The second element is a set of inference rules. These rules are used to deduce the services needed to respond to predicates. Let us take the example of the third predicate "Pressure exceeded threshold", as soon as the engine receives it, it passes by the rules of inferences and it will thus return all the pressure sensors that exist and the methods necessary to identify the thresholds.

2. OBSERVATION CHANNEL (OC) created by the Observation Service for each predicate. Following the reasoning, the OS indicates the services needed to respond to the predicate. Then the OS generates an Observation Channel so that it is responsible for the assembly of the services and thus the calculation of the predicate. Once the string has finished the calculation, it sends the Boolean response to the CM to begin the calculation of the situation.

Advantages of the model:

1. The semantic alignment makes it possible to directly generate or re-use the observation channels for the same predicates expressed in a different way instead of repeating the reasoning.

2. Our model, unlike that of continuum based on a KB, separates well the reasoning (establishment of the CO) from the observation of the context (calculation of the predicates with the data of the sensors).

3. The language dependency between reasoning and observation has been removed.

5.2. Implementation Model

In this part, we will describe the model from the implementation point of view as shown in **Figure 4**.

The Observation Service contains:

1. Cache system connected to the system database.

• It stores each predicate P with the result of the semantic alignment associated with it and the result of the assembly of services that it needs.

It thus keeps the triple (P, P', Assembly of Services).

• It is responsible for verifying the existence of the services in the environment via the database since it is connected to the observatory.

2. The observatory has a major role to maintain a model of the world, *i.e.*, in our case, the list of available services and their descriptions. It is also responsible for automatically updating the reasoning engine observation database and the

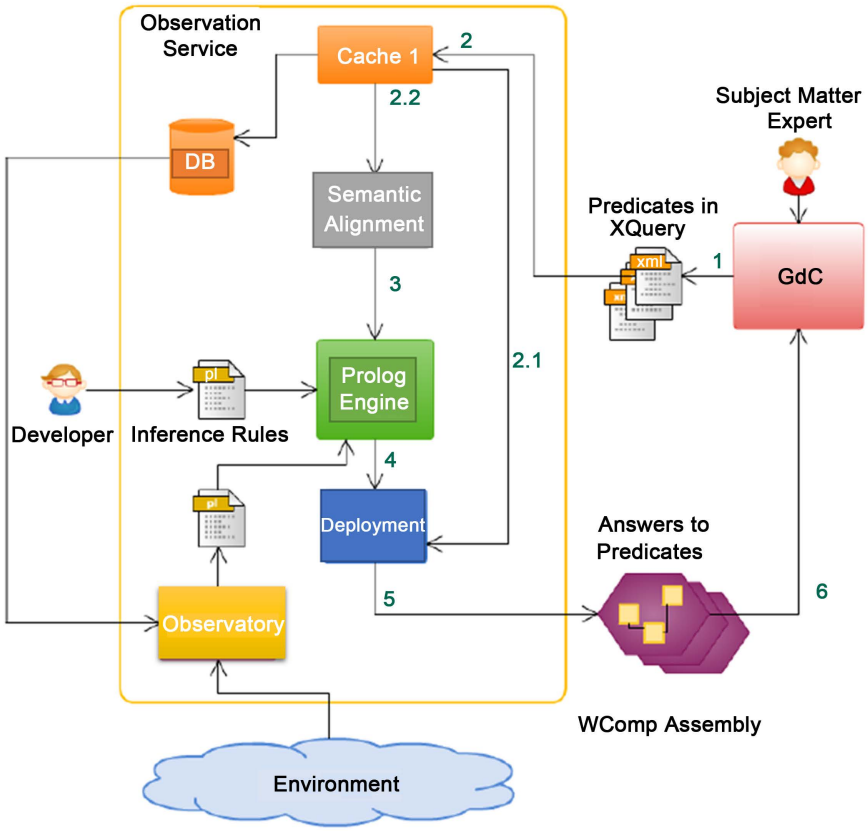

Figure 4. Implementation model.

system database when a new service appears or disappears.

3. Semantic alignment responsible for finding the equivalence of predicate expressions. As this issue is outside our scope, we have not studied how to implement it.

4. Reasoning engine made in Prolog which seemed to us the most interesting because it allows deducing the services necessary to answer the predicates without the intervention of the user.

This engine contains mainly:

• Observation Base that contains the services currently present in the environment. Its main role is to observe new services that appear or disappear so that it is updated automatically.

• Inference rules defined by the developer to describe the relationships between predicates and services in the environment.

5. Deployment responsible of generating the observation channel from the results of the engine. It will instantiate the container WComp which will be our channel of observation. WComp [23] is a prototyping and dynamic execution environment on which the Auto-Adaptive system is based. WComp is composed of an application container which manages components (like beans). This components composition describes the application and can be modified dynamically to adapt the application. Other tools are added to complete the WComp framework (some tools to detect services and devices available in the environment, some

others to adapt the application according to rules and observations on the environment. In our case, we use WComp as an Auto Adaptive System (**Figure 3**) but also as support to implement our Observation Channels. The use of WComp is not an obligation for either of these uses. Another platform could be used to replace WComp. The choice of this platform is only due to the technical mastery of it and the ease of interfacing with it.

6. Observation Channel which corresponds to a container WComp created by the SO in 2 cases:

• After the SO completes the reasoning and provides the necessary names and methods.

• If the predicate was found in the cache system, then the SO will directly create the container without going through the reasoning engine.

This channel has the role of calculating the predicate to respond to the CM. It assembles the services to generate the boolean response. Hence we have N channels of observations for N predicates.

Advantage:

1. Time saving

We believe that the cache system will save us time. In fact, if we assume that:

• $P1$ is the probability of finding the predicate in the cache.

• $P2$ is the probability of needing to reason.

• T_C is the time spent in the cache.

• T_{AS} is the time spent in semantic alignment.

• T_M is the time spent in the reasoning engine.

• T_D is the deployment time needed to generate the WComp container.

We'll have gain in time as per (5).

$$T_{AS} + T_M + T_D > \left(P_1 T_C + P_2 \left(T_C + T_{AS} + T_M\right)\right) T_D. \quad (5)$$

Knowing that Tc is a small time compared to T_M because the cache searches in a list (order of n) and verifies the presence of services associated with the predicate while the engine can make a reasoning that takes a considerable amount of time. In fact, T_M depends on the complexity of the predicate.

However, it remains to verify whether we have a significant time saving in cases where we have only one CM. In fact, once a predicate is analyzed by the SO and its corresponding channel of observation is deployed, there will mostly be no need to repeat this operation.

In spite of this, if we have several applications, each having their own CM that interrogates a single SO, the interest of a cache system is reinforced. The probability of the SO must deploy the same predicate increases. This saves time with a cache system.

2. The system allows high availability since if the SO becomes unavailable, the CM can continue to use the OCs already in place to calculate some of its predicates.

Work sequence:

In below part, we explain how the system works. We have two possible paths:

1. CM sends predicate P to SO.

2. The cache system receives it and verifies initially that the assembly of services associated with the predicate P exists.

2.1. If yes, the cache communicates with the BD to verify the existence of the services in the environment. If this is the case, the cache sends the assembly directly to Deployment to create the container that will then send the response to the CM. If the verification has failed, proceed to Step 2.2.

2.2. If not, the cache sends P to the semantic alignment.

3. The result of the semantic alignment produces P' and sends it to the Prolog engine.

4. When the assembly of components corresponding to the predicate P' has finished being computed in the engine of Prolog, it is transmitted to the deployment service.

5. The deployment creates the WComp container and instantiates in it the components and connectors corresponding to the assembly of the predicate P'.

6. The container can now calculate the predicate P' and transmit the result to the CM.

5.3. Proof of Concept

During our research, we developed a proof of concept to answer a simple predicate the 3rd predicate of the scenario which was "Pressure exceeded Thresholds". To better explain our evidence, we would like to follow a bottom-up approach. We will thus begin by zooming on the WComp container responsible for calculating the predicate to send the response to the CM. Then we will detail each block implemented.

1) WComp container

WComp in general contains a library of ready-to-use components. Each of these components issues events to inform others of an internal state change or to invoke a method of another component. These components are called "Beans". In addition, it is possible to add new beans in WComp or new services in the environment called "UPnP Proxies". These are the representatives of the services in the container. So thanks to beans and proxies, we can do the assembly of services with the methods that suit us to calculate the response of the predicate.

In our case, to respond to $P3$, we need to assemble the service of the pressure sensor with a bean called "Threshold" and to link these two components with an "is Reached" method that will return a Boolean value to CM to determine if the sensor has exceeded the threshold.

2) Observatory

To observe the changes in the environment, we use the UPnP Wizard Designer which is one of the essential bricks of the WComp middleware. In fact, this brick allows the automatic generation or degeneration of proxies for the services that appear or disappear in the environment at the time of execution. We also added a bean to update the Prolog database as well as the cache system.

3) Deployment

This block is made in C # or the WComp container is compatible with this

programming language. It integrates the libraries of SWI Prolog and UPnP open Tools so that it can act as the bridge between the reasoning engine and the chain of observation.

4) The reasoning engine in Prolog

To do this, we began by studying the language of Prolog as well as these two major components: the observation base and the rules of inference.

• Rules of inferences:

We have developed generic rules that deduce the beans and services needed for predicates in the following form:

1. Threshold Reached (name of the sensor) typically for our example it is a pressure sensor. This rule associates the threshold bean with the service found in the observation base with a name identical to the name of the sensor. And so we do a matching to find the suitable service.

2. Is Bigger (service 1, service 2), the aim being to use comparison operators (≤, ≥, !, <, >, =) with service values. We also find a rule for each operator that returns the service names and comparisons beans as well as their methods.

3. And (service 1, service 2), the aim being to use the logical operators (and, or, no and, xor ...). Similarly we have one rule per operator that returns the names of services and comparisons beans as well as their methods.

• The observation base is written in prolog by the observatory. It contains a bean that updates the facts when changing services in the environment.

5) The database

It can correspond to any type of database. A strong point of our system is that it is independent of the technology of the database. To make it simple, we implemented our database tables in C #.

6) The CM

It was provided by the team. We have made some modifications to adapt it to our system so that it sends the predicates to X-query.

7) Semantic Cache and Alignment System

We did not handle them in our proof of concept as it's out of our scope.

6. Conclusions

The objective of our research was to propose a model of a Context-Aware Middleware. In an environment undergoing frequent changes due to the mobility of devices and users, it becomes increasingly difficult to adapt the application at runtime, hence, the need to dynamically discover the environment (context) and to self-adapt the application to these new contextual conditions.

The study of the CONTINUUM project aimed at ensuring continuity of service in ambient intelligence led us to identify some limitations such as the fact that it relied on a Knowledge Base which did both the reasoning and the Observation of the context. Moreover, there is a language dependency between the reasoning and the observation.

So we started the research with these limitations in mind. We started by studying the predicates and their languages and we found that the predicates written

in X-Query corresponded perfectly to our criteria. Then we moved on to another objective, which was to create and automate independent observation channels. To do this, we have separated the reasoning of the observation such that the reasoning is included in an observation service responsible for making the semantic alignment as well as generating the lists of services needed to respond to the predicate while the observation is represented by observation channels or WComp containers generated by the SO. To optimize the system, we have implemented a cache system responsible for verifying the existence of services associated with predicates. Finally, we have developed a proof of concept based on the scenario explained earlier in 2.2. However, further development and testing are still to be performed to accurately evaluate our middleware.

References

[1] IDC (2015) Worldwide Internet of Things Forecast, 2015-2020. Doc No. 256397.

[2] Weiser, M. (1991) The Computer for the Twenty-First Century. *Scientific American*, **265**, 94-104. https://doi.org/10.1038/scientificamerican0991-94

[3] Yang, K., Wang, J., Bao, L., Ding, M., Wang, J. and Wang, Y. (2016) Towards Future Situation-Awareness: A Conceptual Middleware Framework for Opportunistic Situation Identification. *Proceedings of the* 12*th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, Malta, 13-17 November 2016, 95-101. https://doi.org/10.1145/2988272.2990291

[4] Kim, H., Robiul Hoque, M., Seo, H. and Yang, S.-H. (2016) Development of Middleware Architecture to Realize Context-Aware Service in Smart Home Environment. *Computer Science and Information Systems*, **13**, 427-452. https://doi.org/10.2298/CSIS150701010H

[5] Li, X., Eckert, M., Martinez, J.-F. and Rubio, G. (2015) Context Aware Middleware Architectures: Survey and Challenges. *Sensors*, **15**, 20570-20607. https://doi.org/10.3390/s150820570

[6] Khaddar, M.A.E., Chraibi, M. and Harroud, H. (2015) A Policy-Based Middleware for Context-Aware Pervasive Computing. *International Journal of Pervasive Computing and Communications*, **11**, 43-68. https://doi.org/10.1108/IJPCC-07-2014-0039

[7] Perera, C., Zaslavsky, A., Christen, P. and Georgakopoulos, D. (2012) Ca4iot: Context Awareness for Internet of Things. *Proceedings of the* 2012 *IEEE International Conference on Green Computing and Communications*, Besançon, 20-23 November 2012, 775-782. https://doi.org/10.1109/GreenCom.2012.128

[8] Firner, B., Moore, R.S., Howard, R., Martin, R.P. and Zhang, Y.Y. (2011) Poster: Smart Buildings, Sensor Networks, and the Internet of Things. *Proceedings of the* 9*th ACM Conference on Embedded Networked Sensor Systems*, Seattle, 1-4 November 2011, 337-338.

[9] Rey, G., Tigli, J.-Y., Lavirotte, S., Ferry, N., Fathallah, S., Coutaz, J., Fontaine, E., Jouanot, F., Benyelloul, A., Rousset, M., Renevier, P., Pinna-Dery, A. and Hourdin, V. (2010) Modélisation du contexte et Adaptation dans CONTINUUM.

[10] Rasmussen, J. (1983) Skills, Rules, and Knowledge: Signals, Signs, and Symbols, and Other Distinctions in Human Performance Models. *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-13**, 257-266. https://doi.org/10.1109/TSMC.1983.6313160

[11] Rey, G. (2005) Contexte en Interaction Homme-Machine: Le contexteur. Commu-

nication Langagière et Interaction Personne-Système. Fédération IMAG, Université Joseph Fourier.

[12] Rey, G., Tigli, J.-Y., Lavirotte, S., Ferry, N., Fathallah, S., Coutaz, J., Fontaine, E., Jouanot, F., Benyelloul, A., Rousset, M., Renevier, P., Pinna-Dery, A. and Hourdin, V. (2010) Modélisation du contexte et Adaptation dans CONTINUUM. D2.1 and D2.2. https://continuum.unice.fr/_media/livrables:t2.1_2.2_modelisation_du_contexte_et_adaptation.pdf

[13] Jouanot, F., Rousset, M., Morin, G., Johar, A. and Benyelloul, A. (2012) Conquer, gérer l'hétérogénéité. D3.2 and D3.3.

[14] Synyukov, L., Russo, A., Colombi, T., Lavirotte, S., Rey, G. and Bourgeois, H. (2012) Retour d'évaluation terrain. D6.1 and D6.2.

[15] Heer, J., Newberger, A., Beckmann, C. and Hong, J.I. (2003) Liquid: Context-Aware Distributed Queries. 5*th International Conference*, Seattle, 12-15 October 2003, 140-148. https://doi.org/10.1007/978-3-540-39653-6_11

[16] Haghighi, P., Zaslavsky, A. and Krishnaswamy, S. (2006) An Evaluation of Query Languages for Context-Aware Computing. 17*th International Workshop on Database and Expert Systems Applications*, Krakow, 4-8 September 2006, 455-462. https://doi.org/10.1109/dexa.2006.25

[17] Meena, B., Jain, N. and Singh, C. (2013) A Heterogeneous Middleware Architecture for Wireless Sensor Network. *International Journal of Applied Information Systems*, **5**, 36-41.

[18] Gu, T., Wang, X., Pung, H. and Zhang, D. (2004) An Ontology-Based Context Model in Intelligent Environments.

[19] Polyviou, S., Evripidou, P. and Samaras, G. (2004) Context-Aware Queries Using Query by Browsing and Chiromancer.

[20] Yea, J., Dobsona, S. and McKeever, S. (2012) Situation Identification Techniques in Pervasive Computing: A Review. *Pervasive and Mobile Computing*, **8**, 36-66. https://doi.org/10.1016/j.pmcj.2011.01.004

[21] Ranganathan, A. and Campbell, R.H. (2003) A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. *Proceedings of the ACM/IFIP/USENIX* 2003 *International Conference on Middleware*, Rio de Janeiro, 16-20 June 2003, 143-161. https://doi.org/10.1007/3-540-44892-6_8

[22] Kofod-Petersen, A. and Aamodt, A. (2006) Contextualised Ambient Intelligence Through Case-Based Reasoning. 8*th European Conference*, Fethiye, 4-7 September 2006, 211-225. https://doi.org/10.1007/11805816_17

[23] http://www.wcomp.fr/