Augmented Agents: Contextual Perception and Planning for BDI architectures

Arthur Casals, Amal El Fallah-Seghrouchni, Anarosa Brandão

Augmented Agents: Contextual Perception and Planning for BDI architectures

Arthur Casals[1], Amal El Fallah Seghrouchni[2], and Anarosa A. F. Brandão[1]

[1] Laboratório de Técnicas Inteligentes - EP/USP
Av. Prof. Luciano Gualberto, 158 - trav. 3 - 05508-900 - São Paulo - SP - Brazil
{arthur.casals,anarosa.brandao}@usp.br
[2] Sorbonne Universités, UPMC Université Paris 06, CNRS, LIP6 UMR 7606
4 place Jussieu, 75005 Paris - France
amal.elfallah@lip6.fr

Abstract. Context-aware systems are capable of perceiving the physical environment where they are deployed and adapt their behavior, depending on the available information and how it is processed. Ambient Intelligence (AmI) represents context-aware environments that react and respond to the requirements of people. While different models can be used to implement adaptive context-aware systems, BDI multiagent systems are especially suitable for that, due to their belief-based reasoning. Different BDI architectures, however, use different reasoning processes, therefore providing different adaptability levels. In each architecture, contextual information is adherent to a specific belief structure, and the context-related capabilities may vary. We propose a framework that can be used by BDI agents in a multi-architecture scenario in order to modularly acquire context-aware capabilities, such as learning, additional reasoning abilities, and interoperability. When this framework is combined with an existing BDI agent, the result is an augmented agent.

Keywords: context-aware systems, AmI, multiagent systems, BDI, contextual planning, learning

1 Introduction

Ambient Intelligence (AmI) [1] is a term originally created by the European Commission in 2001 [2] and represents the merging between physical environments and information technology, where embedded electronic devices can perceive and respond to the presence of people. When electronic devices or systems capture and use information on the surrounding environment to perform their functions, the interactions between these systems and the individuals present in the environment can be modified and refined in order to adapt and respond in a specific manner. The adaptability level attained by these systems and devices depend on how the surrounding environment information is collected, and the information collection process usually involves different technologies: electronic sensors (temperature, etc.), wireless networks, and human-centered interfaces are among them.

There are a few aspects, however, that need to be addressed to make this adaptability possible. One of these aspects refers to the information structure: it is necessary that information from the environment can be described and structured in order to be used by the adaptability process involved. Devices and systems capable of capturing this information and use it to adapt their functions accordingly are called context-aware systems [3], while information from the environment itself can be referred to as context [4]. Context can be represented and used in different ways – usually depending on which information dimensions or aspects are relevant to the context-aware system or device using it [5, 6]. Different systems, however, present different data needs - both in terms of structure and relevance. Another aspect is the level of complexity involved when these systems interact among themselves. The addition or subtraction of another context-aware system can trigger a certain level of cooperation, which impacts the adaptability process.

Agent architectures are among the ones that can be used by context-aware systems [7]. The belief-desire-intention (BDI) architecture [8] is of particular interest due to its inherent use of contextual information. In fact, the context about the environment in which a BDI agent is situated is represented in its beliefs. Beliefs are used to determine its intentions - what the agent has chosen to do, and how committed it is to that choice [9]. However, different BDI architectures use different belief structures - therefore, translating context into beliefs is a problem highly dependent on the agent's internal architecture. Thus, deploying agents implemented according to different BDI architectures into the same environment can become a challenging problem.

Deploying multiple BDI architectures into multiple environment present challenges related to all interactions between the agents and the environments. Establishing communication between different agents can be achieved through the use of a common communication protocol[1]. Nevertheless, existing agents deployed to new environments may require changes in its belief structure in order to process the new context structure. Adapting an existing agent to a new environment is not only a matter of abstracting the available information, but also making sense of it - which also impacts the planning process used by the agent. When the same agents are used across different environments, multiple abstractions are required, resulting in a scalability problem.

Adding new functionalities to one specific agent could also require modifications in the other agents, as well as in the related environments (i.e., deploying coordination systems). Functionalities such as collective learning, experience sharing, or context-based planning mechanisms such as CPS-L [10] demand even more complexity to be added to the integration model used by the different agents.

With these considerations in mind, we propose a framework that can be used by different BDI agent architectures in order to augment them with context-aware capabilities. The objective of this framework is to serve as an initial step towards solving the problem of creating a multi-BDI environment that can be

[1] http://www.fipa.org/specs/fipa00001/

used in AmI scenarios. The initial version of this framework consists of two modules. The first module addresses the considerations regarding contextual information, augmenting the agent with context-aware capabilities through the use of multiple information sources and structures. The second module (CPS) is an adaptation of the aforementioned CPS-L, which makes the agent capable not only of learning from its previous experiences, but of sharing these experiences among other agents. When these modules are combined with an existing BDI agent, the result is an augmented agent, with modular and inter-operable context-aware and learning capabilities. In the next sections we will refer to BDI agents simply as "agents".

It is important to mention that this framework does not replace or externalize the belief reasoning process. Depending on the deployment environment, context information may not be available all the time. In this case, the use of a context module may relieve the agent from continuously monitoring the environment. Context can be cached, preprocessed, and delivered to the agent, ready to be used in its reasoning process. Similarly, the CPS module makes the planning process more efficient by selecting in advance which plans are feasible. Agents are treated as black boxes, and the framework facilitates the integration between different contexts and agent architectures.

This paper is organized as follows: Section 2 details the general considerations used when constructing the proposed model. The augmented agent model is presented in Section 3, and the framework constructed from this model is detailed in section 4. In Section 5, we present the augmented agent implementation. Section 6 describes the application of a proof-of-concept in order to illustrate our work. Discussions and related work on the modeling and use of AmI agents and systems is presented in Section 7. In Section 8 we present our conclusions on the proposed model.

2 General considerations

Since the framework is intended to be used in conjunction with different BDI architectures deployed into context-aware scenarios, we focused our efforts on understanding the origins, limitations, and aspects related to the processing of the contextual information before it is used by the agent. The process of physically gathering contextual information is not part of the scope of this work, nor is comparing existing BDI architecture implementations. We divided our considerations into (i) the context itself (the presentation and relevance of contextual information available); (ii) processing contextual information before delivering it to the agent; and (iii) the BDI agent planning process, since one of the proposed modules is related to it.

2.1 Context

Contextual information can be collected and distributed differently. It can also be detailed and organized in different levels, depending on its intended use.

Different constraints can also determine its distribution model, such as interoperability with a preexistent communication model or bandwidth limitations. Therefore, the process of collecting and distributing context data can be broken into (i) *Gathering* (what is measured and how it is done), (ii) *Aggregation* (consolidation of collected data into information), (iii) *Representation* (structure used to represent the information), and (iv) *Transmission* (protocols and data contracts used in communication).

Aggregation is the most relevant aspect related to context, since information can be organized in different ways, depending on its purpose. This organization can also differ across different information dimensions. Mobile devices, for example, use different sensors to gather data mostly related to physical environment aspects, such as localization and acceleration. Information on the social information dimension is limited or non-existent. On the other hand, identification cards can retain organizational data – such as role in the company, unique identification record, and clearance level. In this case, the social information dimension is more detailed than in the previous one, while the physical information dimension is almost non-existent.

Different information dimensions can be used in different cases, mostly depending on what is being captured by which sensors. As a term, "information domain" is broadly used to refer to different aspects and purposes of information organization [11]. Generally speaking, information domains can be used to represent deterministic sets of information that are different among themselves in both content and organization [12]. Depending on how the information is organized, the content - or what is being represented - can be determined by its own representation. An ontology, for example, can be defined as a set of terms of interest in an information domain, along with the relationships among these terms [13].

In the scope of this work, context can be comprised of different information domains. Sensor data gathered and aggregated by an internal sensor network, for example, can be considered as an information domain within a given environment. Another information domain could be represented by user preferences stored and organized in a mobile device. When the user is in the environment, the contextual information is composed by both information domains - which can be used by an agent in its reasoning process.

Since our ultimate goal is to simplify the use of context by agents, it is important that we can be able to separate the information into different, treatable sets. These sets must be identified - in order to be distinguishable among themselves - and have a clear structure representation of the information they contain. Representing the information involves not only the data structure used, but also how the information is described. Various expressions can be used by different domains to convey information, which requires different processing rules for each domain. Data structures can also differ across domains. A sensor gateway, for example, can provide information in different formats – e.g. a hierarchically structured database (XML), a single text file or a serialized (binary) structure. A given format, however, doesn't necessarily determine how the information itself

is described: XML files use named elements to separate data, while a text file may require syntactic interpretation.

2.2 Information processing

Occasionally, information processing may be required before the agent can use the contextual information available. It is not uncommon for an agent to use discretized beliefs for its reasoning process. For instance, instead of relying on raw geographical information such as GPS coordinates, an agent may use determined known locations within a limited region (buildings, zones, rooms) on its planning process.

While associating discretized information and parametrized data may seem trivial, it has to be done at some point - and it is necessary to be taken into account in the framework modeling process. As an advantage, isolating the information processing from the agent implementation is beneficial in terms of overall implementation and change management. Different information domains may require different parametrization implementations for the same discretized information set, which may require different effort levels.

Another information process aspect to be taken into account is learning. Despite being able to reason over beliefs related to the environment in which they are situated, not all BDI architectures possess specific mechanisms or functionalities that allow agents to learn from past experiences or to adapt to new, different situations. Our approach towards learning is directly related to the contextual planning system, since the learning process is used as a tool for determining the best course of actions based on previous experiences. In a broader perspective, different learning processes can be used with different sets of experiences.

A specific learning process can be different according to which part of the context is relevant to the actions related to the experiences being processed. In that sense, separating the contextual information translation into beliefs from the agent architecture allows the implementation of learning functionalities associated with different information domains. While fuzzy techniques may be used to determine "hot" and "cold" temperatures, different algorithms may be applied to displacement information in order to establish optimal routes according to associated context conditions, for example.

Generally speaking, processing contextual information before delivering it to the agent means that its belief base is no more a pure reflection of the context as it exists; instead, it is a filtered perception of the surrounding environment. In that sense, it can be seen as a contextual filter to the agent. From the agent's perspective, having such filter allows for abstracting anything other than its own internal reasoning process. From an architectural perspective, this filter allows for other processing modules to use the refined contextual information separately from the agent.

2.3 Planning

The planning process used by a BDI agent is an algorithm that uses the information it possesses in order to generate a sequence of actions. These actions will eventually allow the agent to achieve specific goals (intentions to which the agent is committed). Planning algorithms usually adhere to the following behavior:

- The agent receives information about the environment, which is used to update its own beliefs;
- Desires are updated according to a given criteria (i.e., achievability), and intentions are generated from the updated beliefs and desires;
- A sequence of existing actions (plan) is assembled in order to achieve the generated intentions. The assembly process is part of the reasoning process, but the existing actions must be provided beforehand (usually by the programmer that implemented the agent). Plans can be composed by a single action, which also means that plans can be composed of sub-plans.

After the plans are generated, the agent executes them and observes their consequences (a new environment observation), updating its plans through the same process if necessary.

While different existing BDI implementations are based on this model (such as JADEX and Jason), it can also be modified in order to achieve different goals. In that sense, Chaouche *et al.* [14] proposed a planning management mechanism to be used in conjunction with contextual information in order to optimize the agent's planning process. This mechanism was intended to function as a predictive service, using contextual information in order to verify which actions (among the existing actions known by the agent) are feasible. That would allow the planning mechanism - denominated Contextual Planning System (CPS) - to propose an optimal plan to be executed by the agent.

As we mentioned before, the framework proposed in this work is intended to be used with existing BDI architectures. Because of that, it is not our intention to re-implement the agent's reasoning process, or to replace it with our own reasoning mechanism. However, the CPS mechanism was originally intended to be used in AmI-related scenarios, which involves a high level of context awareness. This is interesting from the context-aware perspective, since the mechanism uses contextual information to determine the feasibility of the agent's actions. With that in mind, we decided to adapt this mechanism into our framework to be used as an aid to the agent. The CPS framework module will be explained in further detail in the next paragraphs.

3 Augmented agents model

With the aforementioned considerations in mind, we propose the following model for an augmented agent (Figure 1). We adopted a modular design to accommodate different BDI architectures within the proposed model.

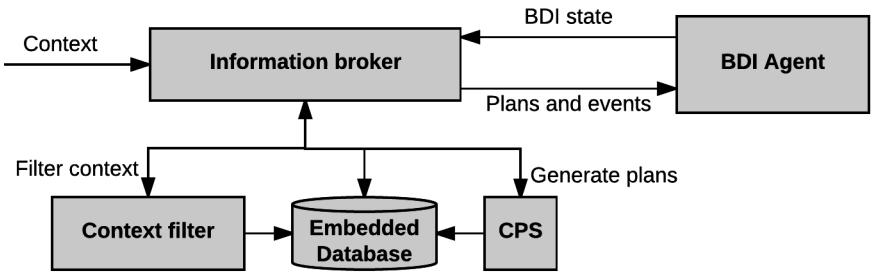

Fig. 1. Augmented agent model in perspective

All contextual information (context) is modeled as a set of different information domains, structured as described before. The module responsible for translating the contextual information received and applying any processing required before delivering it to the agent is called Context Filter. The CPS module, responsible for verify the feasibility of plans for the agent, is also shown. All modules share an embedded database.

The Information Broker is responsible for orchestrating the other modules, as well as retrieving the BDI state from the agent and providing it with the generated plans and the expected events. These events are the result of the context filtering, and they are presented as the agent would expect from an external perception.

This modular design brings benefits to new implementations, such as loose coupling, orchestration, re-usability, and parametrization. Loose coupling is guaranteed through the separation of the framework modules from the agent implementation. While the context processing module can handle several information domains, the processed contextual information is delivered to the agent in a form that it already expects. It also allows for the different modules to be orchestrated through the implementation of a central communication and process coordinator. Also, any module changes or new module implementations remain transparent to the agent.

Using an internal database also facilitates re-usability and parametrization. Specifications related to information domains and learning algorithms can be re-used across existing framework deployments, for example. Additionally, parameters can be modified or transferred to another agent. Historical data can also be persisted and used to further increase the agent's capabilities.

Modularizing and implementing an extensible generic outer layer to be used by different agent architectures is a relatively complex task. While modularizing a software layer and assigning responsibilities to its different components can be done by the use of existing process patterns or established models, implementing new modules within an existing system may be much more challenging. Orchestration schemes must be designed for change, and the information flow must be

flexible enough in order to be altered with minimal effort. As mentioned before, there may also be limitations related to computational costs and deployment environments.

Design requirements were defined to instantiate the proposed model considering the aforementioned aspects. These requirements are listed in Table 1:

Table 1. Design requirements for the proposed model

Requirement	Description
R1	Loose-coupled component modularization within the framework should be used whenever possible.
R2	The framework should fit an internal database to be used by each of the modules in a on-need basis. Any persisted data should be retrievable, modifiable, and shareable.
R3	Different information sources and processing rules should be supported in order to allow for multiple, heterogeneous information domains. The addition of new information domains to an existing process should be as parametrized as possible.
R4	Different learning algorithms should be supported within the CPS. The use of CPS or its learning algorithms should also be optional, since computational constraints on existing systems may prevent its practical use.
R5	Module orchestration should be as simplified as possible. While the addition of new modules may not exempt new code implementation, the inter-module dependency should be kept at a minimal.

Providing an internal database (R2) allows for a higher level of abstraction between the Information Broker and the orchestrated models. The database uses and its relationship with the other modules will be described in the following paragraphs, along with details on each of the modules presented.

The third requirement (R3) involved defining the context structure to be used by the framework. While accounting for different information domains within a context is relatively simple, there's the matter of establishing a structure model for a single information domain. For that purpose - and with the considerations presented in mind - we defined the following structure:

- **Identification:** refers to a local identifier for the information domain, unique for a given environment. While it is important for the agent to distinguish between different information domains, it is reasonable to assume that a global unique identifier may prove difficult to be implemented.
- **Grammar:** represents the language used within the information domain. Due to the considerations above, choosing a grammar over a simple reference matrix for representing the language used within the information domain allows for more flexible information usage scenarios.

- **I/O:** refers to communication metadata associated with the information domain, such as data structure (XML, text, binary) and channel endpoints (REST endpoints, etc.).

In the information domain structure presented above, it is important to notice that identifying communication metadata have certain advantages for the framework as a whole. As mentioned before, processing the contextual information before handling it to the agents is also part of the objective of the proposed framework. In scenarios where computational power or bandwidth is limited, for example, having this information available at a higher level may represent substantial gains in terms of communication efficiency and data processing.

It is also important to mention that in order to abide to the desired level of abstraction, the BDI architecture taken into consideration is as generic as possible [15]. The agent represented in the next section is an adaptation from the BDI architecture model presented in [16].

4 Framework architecture

The next paragraphs detail the proposed framework architecture, along with each of its components. These components, once combined, produce the structure shown in Figure 2. As a framework, its intended structure is meant to be implemented in different programming languages.

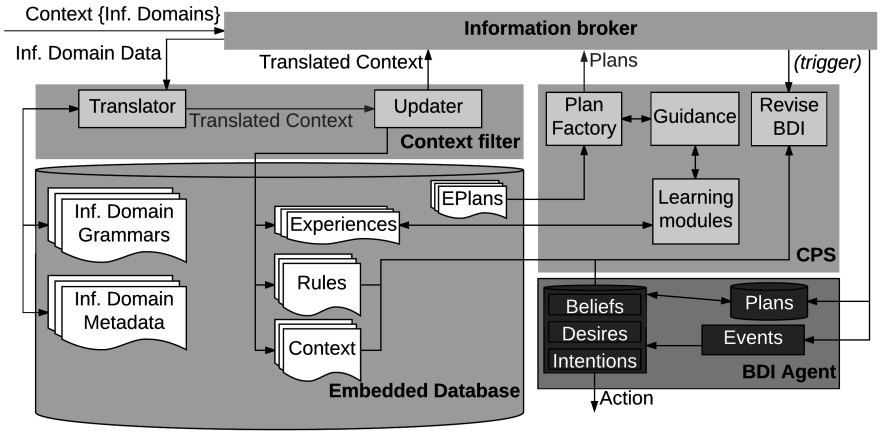

Fig. 2. Augmented framework in detail

4.1 Context filter

The context filter is responsible not only for processing various information domains but also for extracting relevant information to be used by the agent and the other modules. This component is needed in order to capture the contextual information organized according to the information domains structure, thus making it possible for the internal mechanism to process this information accordingly. From a design perspective, the Context Filter responsibilities can be aggregated in two sub-modules, named *Translator* and *Updater*.

All contextual information is delivered by the Information Broker to the *Translator*, which transforms it into a common internal structure that can be used by the system. This process requires the use of information domain mappings, which contain individual grammar definitions that are used by an internal parser. Each information domain is mapped to a specific grammar, describing its structure and the information it possesses. Using a generic parser in association with a parametrized grammar allows for the framework to process new information structures without the need for implementing new code. In other words, new information domains can be added to the framework by the parametrization of new grammars. The result of the process done by the *Translator* sub-module is the translated context.

Once the translated context is produced, the *Updater* sub-module persists it in the database. Different internal structures can be used to store the translated context; it is important, however, that it can be accessed and used by the framework's different modules if necessary. While rules and environment variables such as location and temperature may be persisted immediately, identifying and persisting experiences may require access to historical data. Location information, for example, can be compared to older data to determine the time it took for an agent to move from one place to another.

4.2 CPS

We used the original CPS-L structure as reference when designing this module. CPS-L was originally presented as a planning process method to be used by an agent to select feasible plans from its current set of intentions. These plans are the result of a selection process based on analyzing the current context associated with the agent, as well as its current set of intentions. This process also requires a revision of any restrictions that might be associated with each of the actions allowed to be performed by the same agent.

The CPS-L planning process also incorporates a guidance component that can learn from past experiences. This component uses experiences from previously executed actions to predict the outcome of the analyzed plans, and therefore influencing the planning process.

In addition to the planning process, we also used the original concepts related to plans. The original planning system describes the agent plans as a composition of sub-plans called *Intention plans*. These sub-plans are separated according to

the agent's intentions, in a manner that each of them is dedicated to the achievement of one specific intention. They can be decomposed into several *Elementary plans*, which compose the set of all plans that can be executed by an agent. In that sense, all agent plans can be ultimately decomposed into a subset of the set of *Elementary plans*.

As stated before, our objective was not to propose a new context planning system; instead, our intention is to use it *as-is* in order to demonstrate how different context-related abilities can be incorporated into existing BDI architectures. We present the structure of the CPS module, comprised by the following components: (i) *Plan Factory*, (ii) *EPlans*, and (iii) *Guidance*.

Plan Factory is the component that builds the plan to be executed by the agent. This plan is a composition of *Elementary plans*, built in a manner that different compositions can potentially lead to the same plan result. Additionally, *Elementary plans* can be associated with contextual restrictions, allowing for viability verification according to environment conditions. All *Elementary plans* are stored in a library named *EPlans*, which resides in the embedded database.

Guidance is the component that uses the experiences contained in the database to affect the whole plan generation process. That way, not only the feasibility of a plan can be verified, but also an optimal plan can be chosen among other multiple viable plans. All experiences are associated with an action and a set of contextual variables.

4.3 Information broker

The *Information Broker* module concentrates two main responsibilities: (i) orchestrating the other auxiliary modules, deciding and triggering their use when necessary; and (ii) delivering information (plans and events) to the agent.

Orchestrating the auxiliary modules is done through the use of a mapped information flow - each module is activated when needed, if needed. Delivering processed information to the agent requires knowing how it should receive this information. This is done by using grammars and auxiliary parametrization, making it possible to connect the framework to different agent architectures with minimal effort.

5 Implementation

We implemented a proof-of-concept implementation for the framework using Java. Our programming language choice was based on the availability of existing libraries that could be used by the different modules. In order to maintain its modular nature, each module of the framework was implemented separately. The information flow was done through the use of common interfaces. A simplified UML diagram of the implementation is shown in Figure 3. Other implementation details related to each of the modules and its sub-modules are described below.

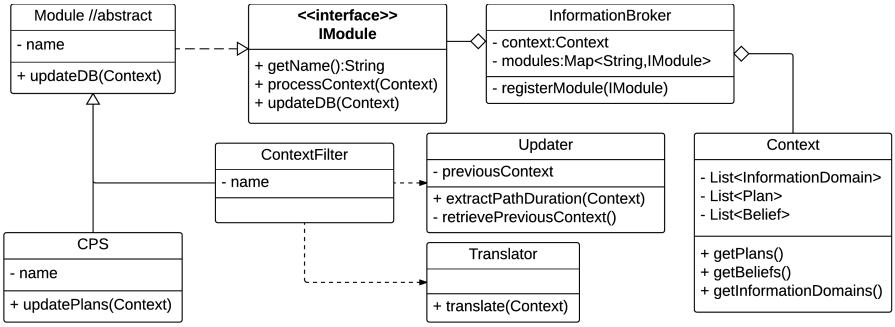

Fig. 3. Simplified UML diagram for the implementation

5.1 Context filter

Although the framework modularity allows for different functionalities to be implemented as modules, the context filter must always be present. This module is responsible for the most part of the data transformations involved in the framework - including persisting data in the database. For this reason, we tried to keep its implementation as parametrized as possible, allowing it to be reused in future work. All sub-modules were also implemented with the same considerations in mind.

Each information domain is represented in the system as a pair of metadata (identification, original data structure) and associated grammar. Therefore, multiple information domains can be registered within the module through the use of parametrization. In order to translate multiple parametrized grammars (representing different information domains) into a common structure we embedded a language parser generator into the *Translator* sub-module, called ANTLR[1]. Since the responsibilities of the Updater sub-module include not only persisting the translated context in the database but also identifying and persisting experiences when required, we used the Active Record pattern [2] in its implementation.

5.2 CPS

We re-factored a previous proof-of-concept related to the original CPS-L when implementing this module. Refactoring this code involved adapting it to use the common information interfaces and the embedded database contained in the framework. It is also important to mention that this code uses a Java-based Prolog engine called tuProlog [3] in its planning process.

[1] http://www.antlr.org
[2] https://www.martinfowler.com/eaaCatalog/activeRecord.html
[3] https://sourceforge.net/projects/tuprolog/

5.3 Embedded database

Centralizing the orchestration of different modules while allowing for them to directly access the embedded database raises concerns on data integrity and database concurrency. In order to minimize these problems and to simplify our implementation, we use an embedded database library called MapDB[1]. This library allows the persistence of Java native collections directly in the memory, while mapping the database into a Java object that is managed by the JVM. Persisted objects can then be accessed as follows:

```
File dbFile = Utils.embeddedDbFile();
DB db = DBMaker.fileDB(dbFile).closeOnJvmShutdown().make();
Locations storedLocations = db.get("locations");
```

5.4 Information broker

As mentioned before, we implemented common information interfaces to be used by all modules. Therefore, every message passing through this module (before reaching the agent) used the same structure. In order to deliver the processed context (events and plans), we had to take into account the data structure used by the agent. Also, to be able to test different agent architectures, we addressed this problem through the use of a structure similar to the one used in the *Translator* sub-module.

6 Application

Having the proof-of-concept implemented, our next step was to test it against an application scenario. We considered a situation involving moving an agent from one place to another, while abiding to a predefined set of restrictions. Two different information domains data structures were used to define the context. The first information domain was used to provide information referencing the location in GPS coordinates. Information about time and temperature was provided by the second information domain. Temperature and time were provided in a comma-separated values format (CSV), while GPS coordinates were associated to identifiers. Each of the information domains were mapped to grammars so the information could be translated by the internal parser. The information on GPS coordinates is shown below, along with its associated grammar:

```
//GPS coordinates as received by the Information Broker:
LAT:-23.557164;LON:-46.730234
//Grammar used by the translator:
grammar CoordinatesID;
coordinates: id COLON latitude SEMICOLON id COLON longitude;
id: TEXT;
```

[1] http://www.mapdb.org

```
latitude: COORDINATE;
longitude: COORDINATE;
COORDINATE: ('-')? [0-9][0-9] '.' [0-9][0-9][0-9][0-9][0-9][0-9] ;
SEMICOLON: ';' ;
COLON: ':' ;
TEXT: [a-zA-Z]+ ;
```

All required parser classes are created by the ANTLR library, allowing the *Translator* to retrieve the relevant contextual information for each information domain:

```
//class CoordinatesIDParser parser generated by ANTLR
String sLatitude = parser.CoordinatesContext.latitude.getText();
Double latitude = Double.parseDouble(sLatitude);
```

In order to discretize GPS coordinates into known locations we used a reverse geocoding process, which returns the nearest known location from a given set of coordinates. We adapted an existing library [1] to make it inter-operable with our embedded database. This process was implemented in the *Updater* sub-module.Using reverse geocoding allowed us to discretize any set of coordinates as follows:

```
//Double latitude, longitude;
String location = storedLocations.findNearest(latitude,longitude);
```

All functionalities necessary to compute experience involving the movement from one location to another (duration, points of origin and destiny, restrictions in place) were also implemented within the *Updater* sub-module. We used only one learning module to process the experiences within the CPS module. The learning algorithm was based on minimal movement duration over the five most recent experiences. Two different agent architectures were used in conjunction with the same framework implementation in order to test its re-usability (plans and events output). Since our goal was to assure that the processed context and plans were being propagated to the agents accordingly, benchmarking the agents was not a part of this work.

The agent architectures used as adjacent structures to the framework were: JASON[2] and BDI4JADE[3]. While JASON uses an AgentSpeak-based syntax to represent plans and beliefs, BDI4JADE is a pure Java implementation of a BDI architecture. This allowed us to test how plans and beliefs could be delivered to different agent architectures.

Our success criteria for this test was based on: (i) correct translation from the information domains data; (ii) correct calculations for the experiences; (iii) successful CPS processing; and (iv) successful delivery of processed context and plans to the two different agent architectures. All of them were met, and the test was concluded with success. A repository containing all the files used as inputs

[1] https://github.com/AReallyGoodName/OfflineReverseGeocode
[2] http://jason.sourceforge.net/
[3] http://www.inf.ufrgs.br/prosoft/bdi4jade/

by the framework are available in a public repository[1], as well as the generated plans and beliefs for both agent architectures.

7 Discussion and related work

The present work aims at serving as an initial step towards solving the problem of deploying different existing agents into multiple environments. The proposed framework accomplishes that by (i) providing means to add extra functionality to existing agents and (ii) implementing a generic contextual translating matrix to be used in conjunction with different vocabularies used by agents.

In terms of implementation, it is worth mentioning that some aspects of the framework may demand different levels of effort from the programmer. Using a parametrized grammar mechanism to properly capture contextual information, for example, is beneficial from the interoperability perspective since it allows different information structures to be used by the framework. On the other hand, modeling and implementing a grammar that properly describes the environment or a specific information domain may not be a trivial task. For that reason, specific technologies or tools may be more or less adequate to implement the proposed framework. The study and comparison of such techniques, however, was not part of the present work.

This framework is not presented as an agent-based modeling tool or another agent architecture. Context processing is presented as a mandatory component model in the framework, since it is required for all subsequent processes. At the same time, we used the CPS module as an illustration of how existing functionalities could be easily adapted and used by other BDI architectures. Enhancing the CPS planning process or proposing an alternative to it were both out of the scope of this work.

Adding extra functionalities to agents is usually done by extending its architecture, bounding the extension to a specific agent programming language [17,18] - a monolithic approach in terms of implementation. While this solution tends to be more efficient, it is not scalable in a scenario where multiple agent architectures must be modified in order to receive a new functionality or capability.

Besides being an extra functionality by its own, context processing is also dealt with in different ways. Dealing with various contexts requires multiple context models, which must be implemented somewhere between the environment *per se* and the deployed agent. Different models may be implemented in different manners, which may lead to interoperability issues when the system is required to process multiple models at the same time. Using information domains is a way to preemptively solve this problem, while maintaining the context modeling process parameterizable enough to facilitate new implementations.

Integrating existing agents with other heterogeneous systems is also a matter solved by the use of a parameterizable translation mechanism. While there are obvious drawbacks in terms of required computational power, new communication modules can be implemented and make use of the translation matrix in

[1] http://gitlab.com/casals/AAF/

place. Using a parser tree generator allows for new domain-specific languages to be implemented through the use of structured grammars, minimizing development efforts.

Providing a framework to support context-aware applications is not a novel idea. Most implementations, however, are bound to specific deployment environments or vocabularies [19–22]. Frameworks such as JCAF [21] and CoBrA [22] are deployed as middleware or publisher/subscriber services, requiring additional infrastructure resources and the use of a centralized communication structure. Being limited to a vocabulary also means that the context is translated into a fixed structure. Using this structure with an existing agent would require another translation process to be implemented. In the case of Intel's Context-Sensing SDK[1], the abstraction layer between the application and the information capturing is also limited, restricted to the sensors available and their information data structures.

Our proposal intends to overcome these concerns. Using a framework attached to the agent instead of relying on a central coordination point simplifies its deployment, and re-configuring the communication flow is no longer necessary. Its modularity also allows the deployment of augmented agents with different capabilities to the same environment. Dividing the context into separated information domains and parameterizing their structures also simplifies the adaptation process that occurs when an environment is subject to changes such as the addition or removal of information sources.

On another perspective, CArtAgO[2] provides a workspace for abstracting the environment that addresses the heterogeneity problem. This concept, however, is structurally different from the model proposed. Our work is in its initial stage, but this difference will become more clear in the future.

It is also important to mention that our application scenario was based on a real-world problem, involving a situation where different logistic constraints are imposed either by a set of rules (opening/closing hours) or restrictions related to different people (agents). While no benchmarking or comparisons were done at this time, it is our intention to use this work as a basis to further experiment on scenarios related to real-world problems, thus providing possible solutions for existing problems.

8 Conclusion and future work

In this paper we presented a framework to add context-aware functionalities when combined with multiple existing BDI architectures. The contribution of this work resides in (i) proposing a context awareness extensibility model to be used in conjunction with multiple BDI architectures and (ii) proposing a context processing mechanism to be used with different context representations.

From a software engineering perspective, the first contribution promotes scalability and re-usability. Parametrization and modularization are used to mini-

[1] https://software.intel.com/en-us/context-sensing-sdk
[2] http://cartago.sourceforge.net/

mize the implementation effort involved in extending an existing BDI architecture. Functionalities already implemented can be re-used as modules attached to different agents, and the parameterization mechanism simplifies the deployment of an augmented agent to different environments. Using an embedded database also provides means to the implementation of more robust functionalities.

The second contribution allows for the simplified capturing and processing of contextual information originated from different sources. We also presented an implementation and subsequent experimentation of the proposed framework in order to properly evaluate its feasibility.

Both the extensibility and the context processing mechanisms were successfully tested within the defined parameters. As we mentioned before, the proof-of-concept implementation presents drawbacks related to the use of available computational resources. The study of these limitations, however, was not part of the scope of this work. Future research will include new functionality modules and the evolution of the framework towards a solution for multi-BDI environments in AmI scenarios, as well as experiments involving the use of different BDI agent architectures and studying the identified limitations.

References

1. De Ruyter, B., Aarts, E.: Ambient intelligence: visualizing the future. In: Proceedings of the working conference on Advanced visual interfaces, ACM (2004) 203–208
2. Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., Burgelman, J.C.: Scenarios for ambient intelligence in 2010. Office for official publications of the European Communities (2001)
3. Hong, J.y., Suh, E.h., Kim, S.J.: Context-aware systems: A literature review and classification. Expert Systems with Applications **36**(4) (2009) 8509–8522
4. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In: International Symposium on Handheld and Ubiquitous Computing, Springer Berlin Heidelberg (1999) 304–307
5. Kim, J., Chung, K.Y.: Ontology-based healthcare context information model to implement ubiquitous environment. Multimedia Tools and Applications **71**(2) (2014) 873–888
6. Nalepa, G.J., Bobek, S.: Rule-based solution for context-aware reasoning on mobile devices. Computer Science and Information Systems **11**(1) (2014) 171–193
7. Kwon, O.B., Sadeh, N.: Applying case-based reasoning and multi-agent intelligent system to context-aware comparative shopping. Decision Support Systems **37**(2) (2004) 199–213
8. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In Allen, J., Fikes, R., Sandewall, E., eds.: Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann publishers Inc.: San Mateo, CA, USA (1991) 473–484
9. Cohen, P.R., Levesque, H.J.: Intention is Choice with Commitment. Artificial Intelligence **42**(2–3) (1990) 213–261
10. Chaouche, A.C., El Fallah-Seghrouchni, A., Ilié, J.M., Saïdouni, D.E.: Improving the contextual selection of BDI plans by incorporating situated experiments. In:

IFIP International Conference on Artificial Intelligence Applications and Innovations, Springer International Publishing (2015) 266–281
11. Hjørland, B.: Domain analysis in information science: eleven approaches–traditional as well as innovative. Journal of documentation **58**(4) (2002) 422–462
12. Hennessy, P.: Information domains in CSCW. Studies in Computer Supported Cooperative Work: Theory, Practice and Design, Eds. JM Bowers and SD Benford, Elsevier (1991)
13. Mena, E., Kashyap, V., Illarramendi, A., Sheth, A.: Domain specific ontologies for semantic information brokering on the global information infrastructure. In: Formal Ontology in Information Systems. Volume 46., Amsterdam: IOS Press, MCB UP Ltd (1998) 269–283
14. Chaouche, A.C., El Fallah-Seghrouchni, A., Ilié, J.M., Saidouni, D.E.: From intentions to plans: A contextual planning guidance. In: Intelligent Distributed Computing VIII. Springer International Publishing (2015) 403–413
15. Wooldridge, M.J.: Reasoning about rational agents. MIT press (2000)
16. Balke, T., Gilbert, N.: How do agents make decisions? a survey. Journal of Artificial Societies and Social Simulation **17**(4) (2014) 13
17. Buford, J., Jakobson, G., Lewis, L.: Extending BDI multi-agent systems with situation management. In: Information Fusion, 2006 9th International Conference on, IEEE (2006) 1–7
18. Singh, D., Sardina, S., Padgham, L.: Extending BDI plan selection to incorporate learning from experience. Robotics and Autonomous Systems **58**(9) (2010) 1067–1075
19. Dey, A.K.: Supporting the construction of context-aware applications. In: Dagstuhl seminar on Ubiquitous Computing. (2001)
20. Gu, T., Pung, H.K., Zhang, D.Q.: A service-oriented middleware for building context-aware services. Journal of Network and computer applications **28**(1) (2005) 1–18
21. Bardram, J.E.: The java context awareness framework (JCAF)–a service infrastructure and programming framework for context-aware applications. In: International Conference on Pervasive Computing, Springer Berlin Heidelberg (2005) 98–115
22. Chen, H., Finin, T., Joshi, A.: A context broker for building smart meeting rooms. In: Proceedings of the Knowledge Representation and Ontology for Autonomous Systems Symposium, 2004 AAAI Spring Symposium. (2004) 53–60