An Agent-Based Architecture for providing Enhanced Communication Services

Romelia Plesa[1], Luigi Logrippo[2,1]

[1]School of Information Technology and Engineering, University of Ottawa, Canada

[2]Département d'informatique et ingénierie, Université du Québec en Outaouais, Canada

{rplesa@site.uottawa.ca, luigi.logrippo@uqo.ca}

Abstract. With the emergence of ubiquitous computing as the new trend in personal communication, there is also a growing need for services to be tailored to the user's specific needs and preferences. We propose an agent-based architecture that allows context-aware communication between users. In seeking a model that is suitable for the design of the required functionalities of our framework, we found significant overlap between the concepts needed in our domain and those used in the Belief-Desire-Intention (BDI) models of agency. The BDI model is one of the most successful theoretical models of rational agents. It provides a convenient terminology and structure for describing such agents, as well as their policies. We present a comprehensive attempt to use the BDI model for describing the architecture and protocols of our context-aware communication system. Our system includes a mechanism for handling conflicting user policies.

Keywords: communication services, personal communications, context-awareness, ubiquitous computing, agent systems, BDI agents, AgentSpeak(L), user policies.

1. Introduction

Among the current trends in personal communication, there is a growing need of communication services tailored to the user's specific needs and preferences. With devices that enable mobile communication becoming more popular, there is an increasing necessity for the users to control and customize their services, making them sensitive to the context in which the communication takes place and influencing the user's availability and reachability for communication.

The literature distinguishes between context-free and context-aware communication systems [1]. *Context-aware* services take advantage of knowledge of real-time context regarding the purpose or the circumstances of an incoming or outgoing call. However, most of today's telephony communication services are *context free*, in the sense that they do not have such knowledge. Context-aware services can be of course much richer and allow the user to manage the use of his communications systems in terms of policies, taking into consideration context information.

Clearly, context-aware systems require much richer architectures than conventional systems. They also require high-level concepts and languages for programming user policies and services. Much work is being done in this area, some of it proposing agent-oriented architectures. This paper presents the results of a study towards an implementation of a Belief-Desire-Intention (BDI) agent architecture for context-aware communications systems. AgentSpeak(L), an agent-oriented programming language, is used to implement the architecture. We show how the BDI approach may be used to provide useful, intelligent services to users in realistic settings. In doing this, we discuss different challenges and issues related to the architecture. The elements of this architecture were presented in [2], with additional details in [3]. This chapter is an extension of [3].

Our architecture is applicable to situations where users have the need to define complex policies on how their communication should be handled, based on information about their current context. Communication requests are handled according to these policies, with the potential of a very high degree of customization.

2. Motivation

The advocates of presence technology and contextual services promise a world where people will be connected *when they want, how they want, and with whom they want* and their communication will be tailored on specific desires and preferences, according to circumstances [4].

One hypothetical scenario, described below, illustrates the capabilities of a system that offers converged services, this means services that combine voice, presence, Web, chat, and other elements. Although sounding futuristic, many individual components have been developed and our work proposes an architecture that will enable the realization of such scenarios.

Bill has a conference call at 9:00 am. It's 8:30 and he is stuck in traffic, but he doesn't worry. He subscribed to a presence management service that knows where he is and how best to reach him. The service forwards the call to his mobile phone so he joins the conference from his car. Once in his office, Bill starts working, but his phone rings continuously, distracting him. A solution is to have a call filtering service that routes all calls, except those from certain clients, to his voice mail. The service recognizes the key callers and routes each one to a personalized message from Bill. Regular callers hear Bill's usual voice message. In the meantime, Bill is able to complete his presentation. He then wants to talk to a customer. He is not sure where the client is (office, lunch etc.) and which network (home, wireline or wireless) he should use. The presence management application will locate the client and will use the appropriate device to call. Bill may also want the information under discussion (the context of the conversation) to be provided to himself and to his client. For himself, the information will be presented via the PC. If the client is in his office, he could receive the information on his PC as well. If he is mobile, then the summarized information could be displayed on his Web-enabled cell phone.

The primarily point being made by this example is that the act of communication is always part of a larger context. Communication services will come to exhibit self-awareness: a sense of why they are being used, of what task is being supported, of the goal to be accomplished.

3. Consolidated Presence Information

Presence information is information concerning a person's online status, availability and reachability across different types of communication channels. Presence is defined [5] as the willingness and ability of a user to communicate with others on the network. On most applications, presence has been limited to "on-line" and "off-line" indicators. The notion of presence in our work is much broader. We want it to include the physical location of the user (*at home*, *at the office*), call state (*ready and willing to communicate*, *currently on a call*), the role that the user is currently taking or the user's willingness to communicate (*available*, *in a meeting*). It can also include indicators that show if a user is logged into a network and whether that user is active, whether a user's mobile phone is switched on, what network it is on, its cellular location and the preferred medium for communication (*voice*, *IM*, *video*, *e-mail).* In this way, presence information becomes relevant to any means of communication, not only instant messaging and buddy lists (applications that are already using it), but also to a large variety of devices and contexts. The notion of context has been discussed in [6] [7]. According to [8], the context is defined as information that characterizes the situation of an entity (a person, a place, or a physical or computational object). In most of the work in the area of context-aware computing, the need of awareness of the physical environment surrounding a user and his devices was addressed by implementation of location-awareness, for instance based on global positioning. Therefore, beyond location, there are other features of the environment that contribute to context. Human factor related context includes information about the user (e.g. their habits), the user's social environment (co-location with others, social interaction) and the user's task. Context related to physical environment includes location, but also infrastructure and information about surrounding resources for computation and communication. Information such as what the user has done in the past can also be part of the context. Some context information, such as the role of the user, can be manually keyed in by the user, while other information, such as location, time of day, can be easily gathered using sensing devices. Other information, such as the current status of the user, can be gathered from sources such as the calendar of the user or from a meeting attendees list.

A combination of presence information and context information (Figure 1) can be used to build an intelligent picture of a user's current situation, status and accessibility. We call this the **Consolidated Presence Information** or CPI, for the user.

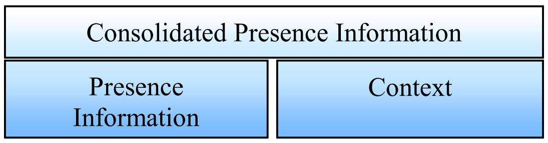


Figure 1 Consolidated Presence Information

"Consolidated Presence Information" means aggregated information from various devices, applications, and attributes such as networks and locations that results into a unified and aggregated view of an individual's current status. This allows users to dynamically set policies that govern the particulars of how users and applications interact with one another and define their preferred device or application for real-time contact. These policies can be based on attributes such as time-of-day, location, or the people attempting to contact them.

For example, a user's policy may describe the behavior required when the following conditions are met: an individual is at work, located in his office, and is writing an e-mail. With no support for aggregation, a combination of database queries must be used in order to determine when these conditions are met. This is unnecessarily complex and is difficult to modify if changes are required. By aggregation, all the information about a given entity is collected and processed and a single query is sufficient to determine if a combination of complex conditions is met.

The processing of these complex policies (which are supported by the architecture presented in this paper) needs knowledge about the users and the environment in which their activities take place, organizational activities etc., as well as complex, intelligent mechanisms for deriving knowledge from the raw information that is available to the system.

Our architecture provides entities responsible for building and maintaining CPI, as well as for storing it. A detailed description of these entities is provided in Section 4. The format in which the information is stored can be, for example, Presence Information Description Format – PDIF [9].

4. Architecture

4.1. Elements of the Architecture

In order to provide complex services that are tailored to users' specific desires and preferences the architectural model needs at least the following functional requirements:

- collection of context information using sensors as well as dissemination of context information, publishing of presence information from users and their devices
- description of user policies and preferences
- preferences-based ubiquitous handling of communication

Figure 2 presents the overall system architecture.

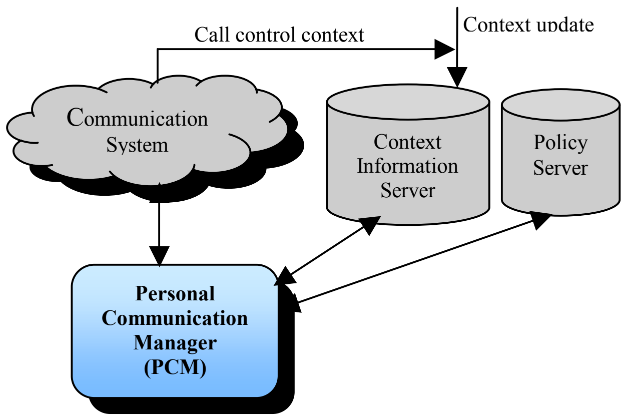


Figure 2 The overall architecture

The communication system is responsible for signaling and communication. The communication part of the architecture is a complex system by itself. Our solution is independent of the underlying communication protocol. It can be based on SIP [10], H.323 [11] or other session protocols. The communication system should provide interoperability between communication protocols of different functionalities.

The requirement that we impose on this architecture is that every message that arrives for a user must be intercepted and dealt with such that user's policies and current context will be considered. We will focus here on the processing of such messages.

The **Context Information Server** (CIS) controls the context updates, stores and distributes the context information. We are not concerned here about how this information arrives. We just assume that there is a mechanism that allows sensors, smart badges etc., to collect and forward it as it changes. The dynamic nature of some context information requires a mechanism for keeping up-to-date information in the Context Information Server, in order to allow services to adapt to the changing context. Among all possible context information, we define a list of significant events (i.e. a user logging into the system, a resource being added/modified, a device becoming available etc.). Whenever an event of this type occurs, triggers are fired that determine the rebuilding of the user's CPI.

The **Policy Server** (PS) manages the user's personal policies, as well as the subscription/ notification policies. The management of policies includes creating, storing, deleting, retrieving and fetching policies for end users. Our proposal for a policy language will be presented later.

We propose the introduction of a new architectural entity, a **Personal Communication Manager (PCM)** that represents each user and is responsible for deciding the flow of actions for the call, based on personal policies and on information about presence and current context. The PCM has access to up-to-date information about the user's context that is used to influence the call functionality. It is responsible for the proper execution of a call. It is the entity that ultimately receives request messages (such as INVITE or SUBSCRIBE for a SIP-based architecture) and decides the actions that should be taken and how the call should be handled. Moreover, it takes into consideration the current context, which is an important part of the decision. The PCM treats relevant events that occur in the system. Such events can be invitations to a call, but can also be updates in presence information or changes in the current context. Based on the event, PCM will decide the next course of action and how the call will be handled. A further role of the PCM is to detect and resolve (or suggest resolutions) for any conflict that arises among policies. The policies are retrieved by the PCM by querying the PS.

The lifetime of a user's PCM starts with the new user being registered to the Presence System and ends with an explicit removal of the user from the Presence System. During this time, the PCM's responsibilities will include managing the calls for the user, based on context and personal policies.

The components of the PCM are shown in Figure 3.

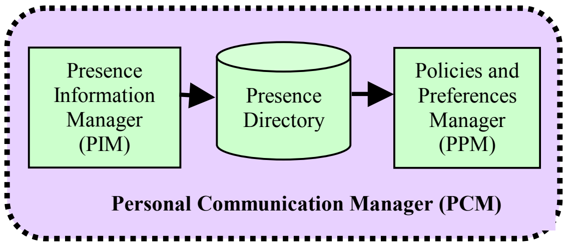

Presence Information Manager (PIM)
Presence Directory
Policies and Preferences Manager (PPM)
Personal Communication Manager (PCM)

Figure 3 Personal Communication Manager

The **Presence Information Manager** (PIM) is the element responsible for building and maintaining the CPI, see Section 3. It does so by aggregating presence and context information from different sources. It manages raw presence data and distils the flow of indicators. The consolidated presence can be maintained by a rule-based process that takes into account presence and context indicators and their ability to reflect the user’s state. Several mechanism have been proposed [12], [13]. The actual mechanism to be used for reasoning about context in our architecture is part of our future work. We would like to provide the agents with a choice of reasoning and learning mechanisms that they can use to derive new information from the raw data provided. These mechanisms can be provided in the form of libraries that the agent can use.

The **Presence Directory** is a repository in which the CPI is deposited and retrieved.

The **Policies and Preferences Manager (PPM)** contains the preference logic and processes that respond to requests to contact an entity. The CPI is interpreted by the PPM to establish the best method for contacting a user at a particular moment, at a given location, based on availability, device capabilities and personal preferences.

4.2. Presence Management Strategy

With this architecture, the process of building the CPI is done in several steps, thus allowing separating the context acquisition and management from reasoning with context knowledge. We can distinguish three layers (Figure 4):

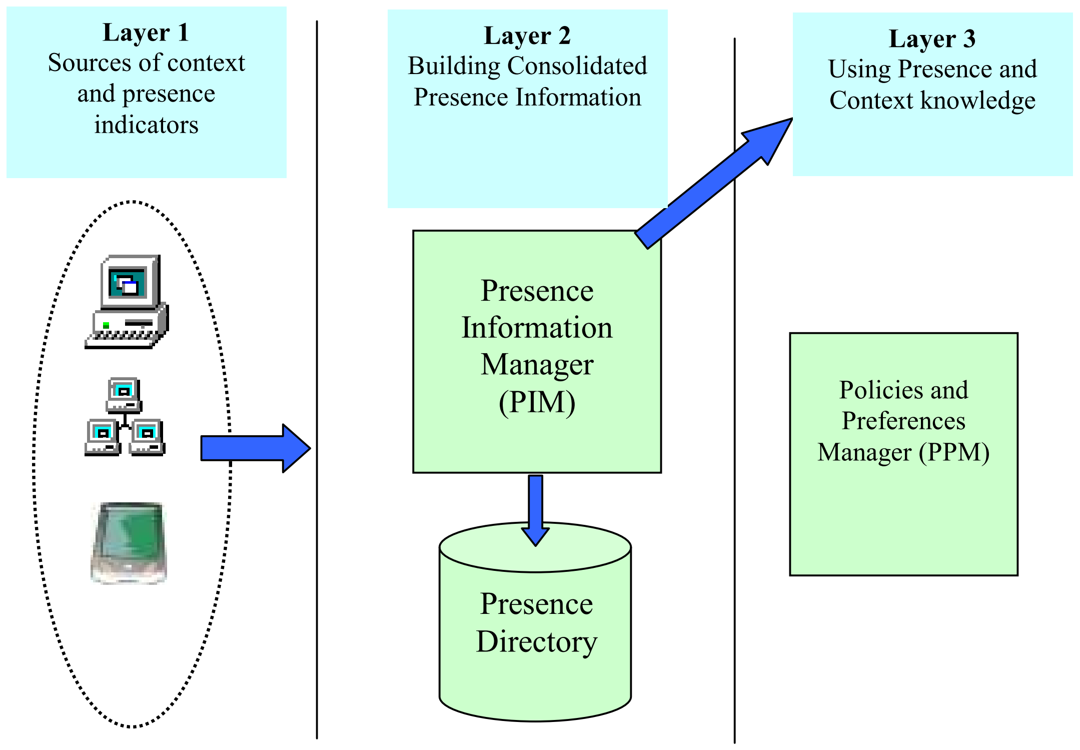

Layer 1
Sources of context and presence indicators
Layer 2
Building Consolidated Presence Information
Layer 3
Using Presence and Context knowledge
Presence Information Manager (PIM)
Presence Directory
Policies and Preferences Manager (PPM)

Figure 4 Layered Management of Information

- sources of context and presence information

- presence management
- users of consolidated presence information

An entity’s presence, the CPI, is constructed from a series of presence and context indicators that come from access networks, directly from the user’s terminals or from third party information sources. These indicators are combined to form a higher-level view of presence for a user. This combination has to be done in real-time so that the entity’s presence can be projected out in advance of any attempt to communicate with the entity.

Presence and context indicators are generated for the PIM from several sources: First, presence information comes as a side effect of a user utilizing the network. For example, a registration on a GSM network when the user’s cell phone is activated constitutes an indicator that the user is currently active in a certain location and ready to use the network. This can generate presence as well as context information. A second source arises directly from presence clients that reside on a user’s terminals (PC, PDA, phone etc.), which will generate presence information. A third source of indicators is from third party services. For example, a hotel registration system can generate an indicator as a side effect of a guest registering upon arrival. This information will be delivered in the CIS and will represent context information about the user. Similarly, an employee logging into the system when he goes to work will constitute an indicator about his location and availability for communication.

Figure 5 shows how the PCM obtains the presence information and the context information. When any change occurs in the presence status of a user, PCM will be informed.

This is possible because, as explained earlier, PCM intercepts the messages exchanged in the communication layer, so it will be aware of any notification of a change in the presence information. In this way, any change in a user's presence will result in the PIM reapplying the rules and rebuilding the user's consolidated presence.

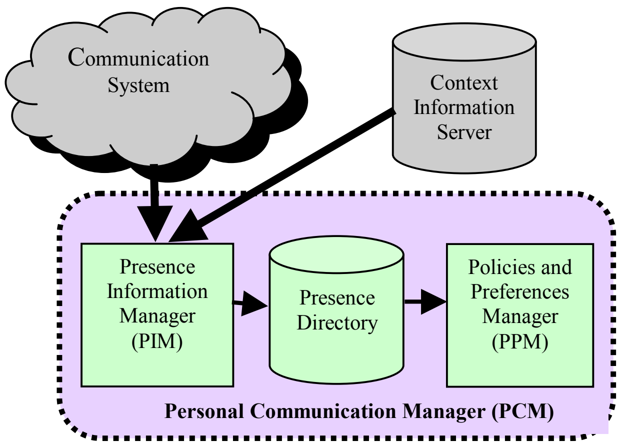

Communication System
Context Information Server
Presence Information Manager (PIM)
Presence Directory
Policies and Preferences Manager (PPM)
Personal Communication Manager (PCM)

Figure 5 Sources of indicators for PCM

In a SIP-based architecture, for example, the entity that deals with the user's presence is the Presence Agent [4]. For such architecture, PIM will be informed by the Presence Agent about the changes that occur in the user's presence. This can be accomplished by allowing the PIM to subscribe to any relevant event that the Presence Agent receives, i.e. any update from PUAs (Presence User Agent). In this way, any change in a user's presence will result in a notification being sent to PIM, which will have to reapply the rules. In this case, it is necessary to define a new event package that will contain the events that generate presence indicators. This event package is to be defined as an extension of the SIP specific event framework.

The context information (which resides in the CIS) is obtained by PIM by querying the CIS, which will fetch the relevant information and send it to the PCM. Having the presence information and the context information, PIM will update the CPI in the Presence Directory.

Figure 6 uses Use Case Maps [14], a popular notation for describing causal scenarios, to show the process of building the network presence for a user.

The process has two triggering points: a change in the presence information or any event that happens in the CIS, i.e. an update in the context information for the user. In case any of these happens, PIM will first obtain presence and context information. After that, it will generate the consolidated presence for the user, which will be deposited into the Presence Directory. Also, some information about the current context may change after this process, so the CIS must be updated as well.

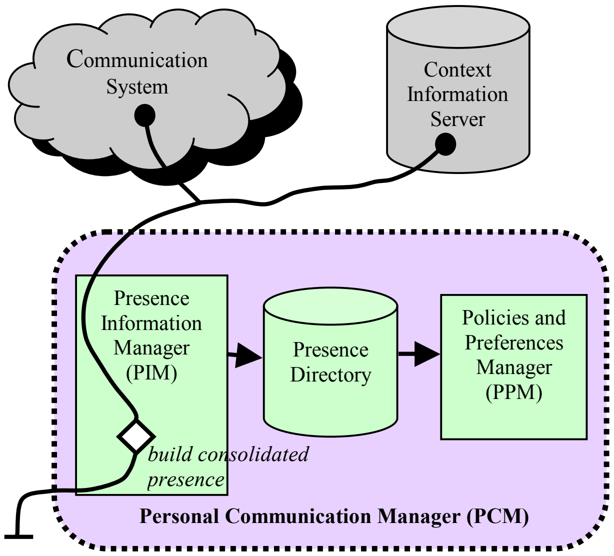

Communication System
Context Information Server
Presence Information Manager (PIM)
Presence Directory
Policies and Preferences Manager (PPM)
build consolidated presence
Personal Communication Manager (PCM)

Figure 6 Consolidating Presence

For simplicity, we used a stub to describe the actions of the PIM. The detailed sequence of responsibilities included in this stub is shown in Figure 7.

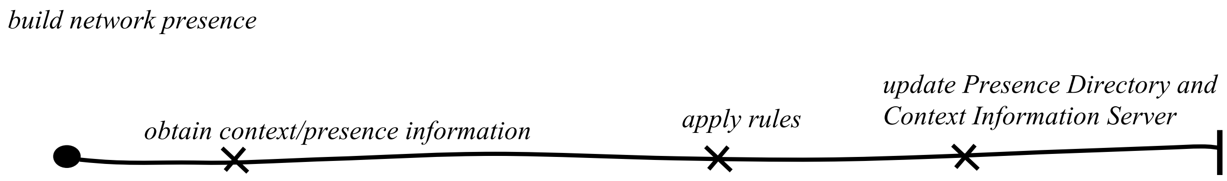

build network presence
obtain context/presence information
apply rules
update Presence Directory and Context Information Server

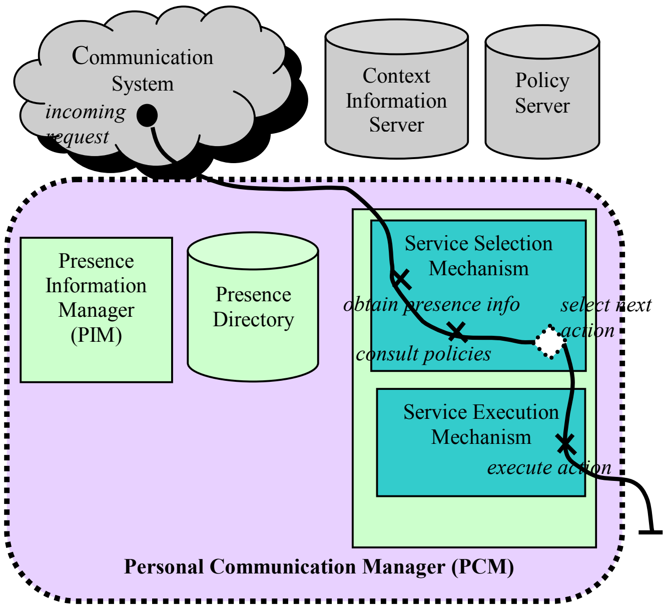
Figure 7 The *"build network presence"* stub

4.3. Context Aware Call Handling

The feature selection mechanism and the feature execution mechanism will be incorporated into the PCM, more precisely in the PPM.

PPM, which contains the preference logic and rule-based processes to respond to different requests, consults the user's personal policies as well as the information stored in the Presence Directory and decides about the handling and execution of the call or of any other request that arrives.

In Figure 8, we show the actions of the PPM using the Use Case Maps notation. The selection of the next action to be executed is a complex mechanism, which can be implemented in different ways. For this reason, we used a dynamic stub to represent it. The action is then executed by the Service Execution Mechanism of the PPM.



Communication System
incoming request
Context Information Server
Policy Server
Presence Information Manager (PIM)
Presence Directory
Service Selection Mechanism
obtain presence info
consult policies
select next action
Service Execution Mechanism
execute action
Personal Communication Manager (PCM)

Figure 8 Call Handling

At the communication level, we assume that there are access points for user's communication devices that manage the sessions involving these devices. Conversation devices such as phones or IM clients, as well as messaging devices such as pagers should be supported.

As decisions on how to reach a user are made, the appropriate device is contacted by performing address lookup using the information that is found in the Directory Service component of the CIS.

There are a number of languages for specifying policies. The Call Processing Language [15] allows users to define how they wish calls to be handled, but it is limited in a number of ways that make it unsuitable for expressing complex policies for call control [1]. LESS [16] inherits the basic structure and many elements from CPL and enhance it with elements for end system services, thus allowing end users to program their own communication services. A policy language targeted to policies in the communication domain has been defined in [17]. We will propose now an approach to define policies based on BDI and AgentSpeak(L).

5. BDI and AgentSpeak(L)

One of the most successful theoretical models of rational agents is the Belief-Desire-Intention (BDI) model [18] [19]. The concept of BDI agents has also proven useful in practical applications, for example, in the defense industry [20] [21].

The BDI framework was developed in the 1980's by Georgeff and Lansky [22]. It has since been implemented in several software platforms, current examples being Jack Intelligent Agents [21] and Jam Agents [23]. The wide appreciation of the BDI model is witnessed by the development of a BDI logic [24], the definition of BDI-based languages (AgentTalk [25], 3APL [26], AgentSpeak(L)[27]) and the creation of BDI-based development tools such as dMARS [29].

The BDI model provides a convenient terminology and structure for describing intelligent agents. Unlike many other agent systems the BDI framework has had many practical applications, by which the theory and terminology have been made clearer and more generic than for other system.

BDI agents have been applied and are suited to model highly dynamic and unpredictable situations. By only partially expanding alternative plans of actions they can remain responsive to changes in system state. They provide ways to recover from failed actions and to customize reasoning for specific situations.

They describe also how to handle conflicting actions and goals, and to modify already executing actions, all of which are needed in dynamic simulations.

Therefore, the BDI framework provides means to specify complex domain-specific behavior. There are numerous published examples that demonstrate complex domain specific behavior being captured and modeled in intelligent agent applications.

In the attempt to bridge the gap between theory and practice, the BDI architecture defines a model that shows a one-to-one correspondence between the model theory, proof theory and the abstract interpreter. The following concepts were developed, as included in the AgentSpeak(L) language.

- An agent's **belief state** is the current state of the agent, which is a model of itself, its environment, and other agents.
- The agent's **desires** are the states that the agent wants to bring about based on its external or internal stimuli. The distinction between desires and goals, while important from a philosophical perspective, is not significant in this context. Henceforth, we shall use the word ***goals*** for desires.
- When an agent commits to a particular set of plans to achieve some goal, these partially instantiated plans (i.e., plans where some variables have taken values) are said to be an **intention** associated with that goal. Thus, intentions are active plans that the agent adopts in an attempt to achieve its goals.

The AgentSpeak(L) programming language was introduced in [27] [28], where additional details on it can be found. It is a natural extension of logic programming for the BDI agent architecture, and provides an elegant abstract framework for programming BDI agents. The behavior of an agent (i.e., its interaction with the environment) is dictated by a program written in AgentSpeak(L). The beliefs, desires, and intentions of an agent are not explicitly represented. Instead these notions are programmed in the language AgentSpeak(L).

At run-time an agent can be viewed as consisting of a set of beliefs, a set of intentions, a set of events and a set of selection functions.

A **belief atom** is simply a first-order predicate in the usual notation, and belief atoms or their negations are termed **belief literals**.

A **goal** is a state of the system, which the agent wants to achieve. AgentSpeak(L) distinguishes two types of goals:

- **achievement goals** - are predicates prefixed with the operator "!".They state that the agent wants to achieve a state of the world where the associated predicate is true.
- **test goals** - are predicates prefixed with the operator'?'. A *test goal* returns a unification for the associated predicate with one of the agent's beliefs; it fails if no unification is found.

A **triggering** defines which events may initiate the execution of a plan. An *event* can be internal, when a subgoal needs to be achieved, or external, when generated from belief updates as a result of changes in the environment. There are two types of triggering events: those related to the *addition* ('+') and *deletion* ('-') of beliefs or goals.

Plans determine the **basic actions** that an agent will perform on its environment. Such actions are also defined as first-order predicates, but with special predicate symbols (called action symbols) used to distinguish them from other predicates. The syntax for a plan in AgentSpeak(L) is:

```
p ::= te : ct <- h
```

A plan is formed by a **triggering event** (`te`), followed by a conjunction of belief literals representing a **context** (`ct`)**.** The context must be a logical consequence of that agent's current beliefs for the plan to be **applicable**. The remainder of the plan is a sequence of basic actions or (sub)goals (`h`) that the agent has to achieve (or test) when the plan, if applicable, is chosen for execution.

Intentions are the plans the agent has chosen for execution (particular courses of actions to which an agent has committed in order to handle certain events). Intentions are executed one step at a time. A step can query or change the beliefs, perform actions on the external world, suspend the execution until a certain condition is met, or submit new goals.

The following example is taken from [27] and shows some AgentSpeak(L) plans.

```
    +concert (A,V) : likes(A) <- !book_tickets(A,V).
    +!book_tickets(A, V) : ¬busy(phone)
         <- call(V);….;!choose seats(A,V).
```

The first plan says that when a concert is announced for artist A at venue V (so that, from perception of the environment, a belief concert(A,V) is *added*), then if this agent likes artist A, then it will have the new goal of booking tickets for that concert. The second plan tells that whenever this agent adopts the goal of booking tickets for A's performance at V, if the telephone is not busy, then it can execute a plan consisting of performing the basic action call(V) (assuming that making a phone call is an atomic action that the agent can perform) followed by a certain protocol for booking tickets (indicated by'. . .'), which in this case ends with the execution of a plan for choosing the seats for the performance.

The AgentSpeak(L) interpreter also requires three *selection functions*:

- *SE* selects a single event from the set of events;
- *SO* selects an "option" (i.e., an applicable plan) from a set of applicable plans
- *SI* selects one particular intention from the set of intentions.

The agents test whether the events generated during belief revision activate any new plan instance. Only one plan can become intended in a single reasoning cycle, and only one intended plan can execute the next part of its body. This can generate new beliefs, goals, or a request for a basic action execution (in the environment) is sent to the interface. New beliefs and goals may serve as triggering events for plans in following reasoning cycles, while actions produce changes in the environment.

Even though there is a body of research on implementing agents on communication devices [29] [30], no attempt has been made, to our knowledge, on using the BDI model to implement a communication system architecture.

6. BDI and Context Aware Communication

In this section, we show how the architectural elements and functionalities that we have identified for the presence system can be mapped on the BDI model.

6.1. BDI Mapping

The selection function of the PPM is its most important function. Its responsibilities are to determine, based on the consolidated presence information available and also based on the user's preferences and policies, the action that should be executed next.

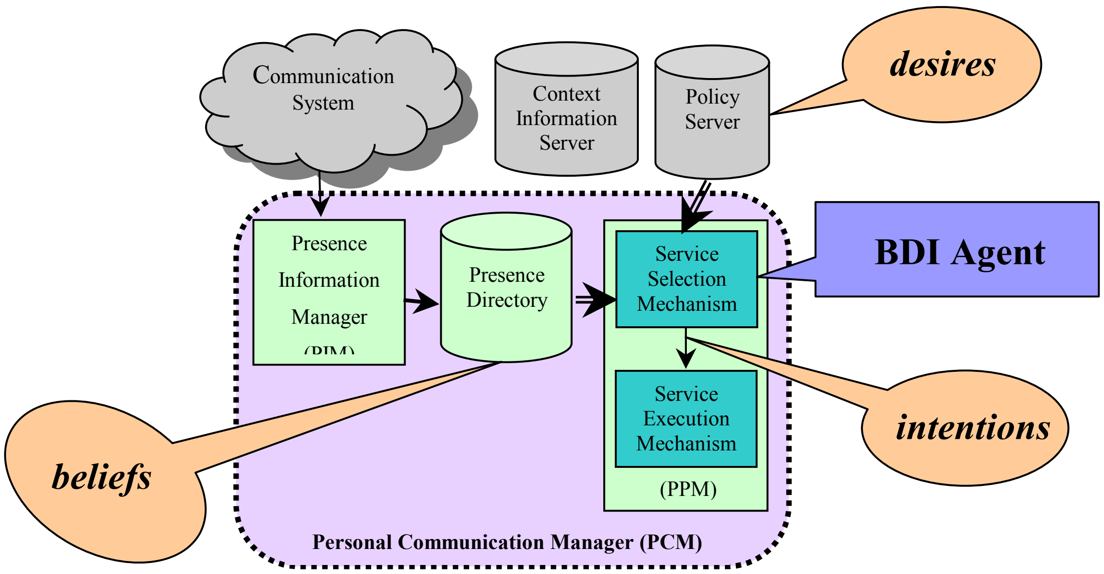

Communication System
Context Information Server
Policy Server
desires
Presence Information Manager (PIM)
Presence Directory
Service Selection Mechanism
BDI Agent
Service Execution Mechanism
intentions
beliefs
(PPM)
Personal Communication Manager (PCM)

Figure 9 BDI mapping

By looking at the BDI model as well as at the PPM component of our PCM and the way we want it to function, we found significant overlap between the requirements of our domain and the concepts discussed in the BDI model of agency.

We consider that the PPM component that we propose as a part of the system's architecture is a BDI agent (Figure 9):

- The *CPI*, stored in the Presence Directory and representing the characteristics of the environment and the user's presence information and which is updated appropriately after each sensing action and each change in a user's status, will represent the *beliefs*.
- The *policies*, which are stored on the PS and seen as objectives to be accomplished, can be considered agent's ***desires*.**
- The output of the selection function, representing the next course of action, will be the agent's *intentions*.

BDI agents fulfill the requirements needed for our framework, providing a way to interleave deliberation with responsiveness and limit the amount of forward deliberation required to act rationally. BDI agents can partially search and expand planned actions allowing them to select good alternatives, while avoiding constant deliberation and its associated time penalty.

6.2. Example

In order to further justify the reasoning behind the adoption of an agent-based approach to support context-aware services and to provide useful, intelligent services to users in realistic settings, we will consider the following situation:

Dr Smith, a doctor at the hospital, needs to achieve the following tasks on a usual day in the hospital:

1. *Arrive to the hospital*
2. *Log into the hospital's system in order to advertise his presence in the hospital*

3. *Attend the patients visits, scheduled every morning, together with other doctors.*
4. *Get the schedule for his consultations and surgeries that he has to perform that day*
5. *Perform his activities, according to his schedule*
6. *Time permitting, being able to assist in any emergency situation that can occur*

Within an agent context, the above tasks may be represented as a set of goals that need to be fulfilled in a given sequence. In order to fulfill each goal, a specific sequence of actions must be executed, i.e. a plan will be executed.

There may be a number of plans for achieving the same task. For example, in order to achieve the second goal, there can be two possible plans: the doctor can log in from the computer in his office, or, in the case that something occurred in his way to the office, he can use his PDA to advertise his presence in the hospital. The plan that will actually be followed will depend on a number of factors, including doctor's preferences, emergency and unexpected situations that can occur and so on. As a result, the agent that represents the doctor has to perform *plan selection*, based on some criteria.

Assuming that the doctor logs in from his office computer, the corresponding plan becomes an intention, meaning that the doctor intends to execute the plan in order to achieve his goal. This might require the execution of another set of sub-tasks, for which, again, there might be a number of plans to achieve them. In this manner, the process can continue, attempting to achieve all goals by executing the appropriate plans, which may trigger other sub-goals, and so on.

In the process of executing the plans, however, a number of problems may arise. For example, suppose that Dr. Smith is not able to log from his computer, due to technical problems. In such case, Dr. Smith's agent should be able to find alternative plans for reaching the goal, for example by suggesting to use another computer. In other words, the agent needs to perform *plan failure recovery*. The new alternative plan must take into account the new context.

Another potential problem the doctor might face is *conflict between different goals*. Suppose, for example, that the doctor wishes to play golf with his friends in the morning, which may conflict with his plan to attend a presentation on the latest news in his area of specialty. The agent should be able to *resolve* this conflict by arranging a different time to play golf, or it might have to perform *goal selection* in order to make a selection about which goal is more important.

It can be seen from the scenario we presented above that as the number of tasks and alternative plans involved increases, the complexity of the reasoning that the agent needs to perform increases significantly. This creates the need of providing automated support for the user, in terms of responding to the user's policies as the context changes. However, particular challenges arise due to the dynamic nature of the environment.

Our proposed solution consists of agents, the PCMs, each representing one user, which will be capable of perform tasks as the ones described above. A PCM has access to the Consolidated Presence Information of the user it represents. With a library of plans, representing the user's policies, the PCM is capable to decide what actions to execute.

With respect to the scenario we have described, the BDI model provides some essential features:

- **Context-awareness.** The choice of plans must take into account the current context in which the user is situated (for example, the physical location or the latest changes in his schedule), as represented in the Consolidated Presence Information. The agent's beliefs base is updated with all the changes in the environment, so that decision can be made based on the real status of the environment.
- **Plan selection.** The agent is able to make a choice between different plans, in case there are a number of alternative plans for achieving the same goal. The choice may depend on the overall cost, the risk factor, the user preferences, etc. Appropriate decision procedures must therefore be supplied for supporting plan selection.

- **Plan failure recovery.** If a plan fails, the agent is able to retract properly and select an alternative plan.
- **Conflict resolution.** If a situation occurs where the user has a number of goals that cannot be achieved simultaneously, the agent must be able to make a decision about which goals to try to achieve, based on the importance of the goals or the costs of executing the plans.

6.3. Proof of Concept

Given the mapping that we have presented in Section 6.2 between the BDI model and the functionalities of our proposed architecture, we have implemented the PCMs as BDI agents. In particular, the agents are programmed in AgentSpeak(L) and we use Jason [27] as the interpreter for AgentSpeak(L).

The BDI mechanism incorporates both the selection mechanism and the execution mechanism that are required by the functionality of the PPM (Figure 9). Decisions on call routing and future actions to be executed are based on individual user's policies, expressed as AgentSpeak(L) plans.

In order to illustrate the essential features that need to be provided by the system and show how the BDI model can provide the required functionality, we implemented a pilot demonstration of the framework described here as the first step of implementing a fully functional simulation model based upon a real communication system enhanced with context services. The demonstration consists of six agents each managing the communication for one user. The users are situated in a hospital setting. Initially, the agents share a set of beliefs, which are stored in a relational database. This represents the CIS component of the architecture (Fig. 1). In addition, each agent can have its own beliefs. Plans, representing user's policies, are defined for each agent in order to cover different scenarios that we envision occurring in the system. The external events to which an agent must respond are request messages that come from the Communication System (Fig. 1). Such messages can be, for example, INVITE or REGISTER requests on a SIP based architecture. In our demonstration of the framework, we simulate these requests (the environment) by feeding the agents, periodically, with events that correspond to real requests.

The plans implemented by the agents will affect the current state of the system. We show that agents will respond to contextual information that comes from the system and will be able to behave accordingly, taking into consideration the policies of the users that are defined as plans. We also want the agents to respond to system changes brought about by other agents. Finally, we show that communicating agents can negotiate in order to avoid possible conflicts.

In order to make sure there are no discrepancies between the agents' understanding of the system and of the actions they can perform on it, we need the assumption of a shared data model among all agents. The data model provides a device-independent description of the world. We defined an *Entity-Relationship data model* that provides the means to hold the context, as well as the semantics to describe the simulation domain. With this, the agent's beliefs will be represented as facts against the data model. The agent's desires, in the form of plans in AgentSpeak(L), are formulated in terms of entities, attributes and relationships contained in the data model. The advantage of using this data model approach is that each entity class in the data model can be viewed as a finite domain, with the object instances themselves as the elements in that domain. The object's attributes can be used to form the basis for specifying constraints and reasoning in terms of the data model.

Each agent has a set of plans that models the user's policies. We will describe in detail the plans for handling a specific situation:

Dr. Smith is at this moment in his office. A colleague calls him from within the hospital. The presence system will decide, based on the doctor's location, that he is available on several devices: his PDA, his cell phone or his desktop phone. Since the caller is using a phone, and the PDA does not have voice capabilities, the choice is narrowed to two devices. Dr. Smith has a policy that designates his cell phone *for calls from his family. Therefore, the presence system decides to transfer the call to Dr. Smith's desktop phone.*

Before we start discussing the plans to realize this policy, it is worth discussing the initial belief base that is required in the running version of the program. The beliefs of the agent are based on the data model that we have defined. Thus, the belief base will contain information about the users in the system, the devices available to the users, the locations and the activities that users are involved in. All these represent the CPI for the user. Relationships between these entities are modeled using databases that contain references to the entities involved in the relationships. (For example, PERSON_PERSON specifies the relationships between two persons identified by their unique ID number. Similarly PERSON_DEVICE specifies the devices associated with a person).

All this information is available for the agents in the Presence Directory component of the architecture. From the agent point of view, the information in the database is mapped into predicates. For example:

```
PERSON(john, user00001, TRUE, available, call, 562-5800, eng, on_the_phone, j,
doctor)
DEVICE(fix_phone, dev00001, ip_phone, mitel, 5010, eng, call, open, 563-2345)
LOCATION(or1, operating_room, canada, ottawa, 345 carling, general hospital)
ACTIVITY(a1, meeting, work_related, 11:00:00)
PERSON_PERSON(PERSONID_1,PERSONID_2,RELATIONSHIP)
```

Agents are able to consult, insert or delete values from this database by simply adding, deleting or querying the facts as beliefs.

This policy is enforced using a plan for the situation in which incoming call arrives, from X, for Dr. Smith. The triggering event for the plan is incoming_call, with a parameter specifying the caller.

```
+incoming_call(X):true <-
      !get_devices(DeviceList);
      !get_relationship(X,Relationship);
```

```
        !get_location(Location);
        !get_activity(Activity);
        !process_call(X, DeviceList, Relationship, Location, Activity).
```

The first thing to be done when a call request arrives is to obtain the list of devices where the doctor can be reached. This is done by adding the subgoal get_devices, which is achieved using the following plans:

```
+!get_devices(DeviceList) : true
<- .findall(X, device(X,Y,Z,_,_,_,"call",_,_), DeviceList); ?name(N);
?person(N,ID,_,_,_,_,_,_,_,_);!get_user_devices(DeviceList,ID,UserDeviceList).
+!get_user_devices(DeviceList,ID,UserDeviceList)                                    <-
!get_u_devices(DeviceList,ID,[],UserDeviceList).
+!get_u_devices([],ID,L,L).
+!get_u_devices([D|T],ID,L0,L) : device(D,Did,_,_,_,_,_,_,_) & person_device(ID,Did)
          <-!get_u_devices(T,ID,[D|L0],L).
+!get_u_devices([D|T],ID,L0,L) <- !get_u_devices(T,ID,L0,L).
```

First, a list of all devices having the capability "call" is obtained by querying the device table. Recall that this is part of the current context for the user, which is stored in the Presence Directory and is available for the agents to be consulted and queried. After that, the subgoal `get_user_devices` determines which devices from this list are associated with the user.

The next thing to be done is to determine the relationship that the caller has with the user. This is done by verifying if there is information in the PERSON_PERSON table in the Presence Directory that contains both the ID of the user and of the caller. In case there is no such information, the relationship is said to be unknown.

```
+!get_relationship(X,R):
          name(N)&person(N,ID,_,_,_,_,_,_,_,_) & person(X,Xid,_,_,_,_,_,_,_,_) &
     person_person(ID,Xid,R)  <- true.
+!get_relationship(X,R) <- R = "unknown".
```

The current location of the user needs to be determined as well. If there is no information about the user's location, the location is set to unknown. In a similar fashion is determined the activity that the user is currently engaged in.

```
+!get_location(LName): name(N)&person(N,ID,_,_,_,_,_,_,_,_) & person_location(ID,L)
     & location(L,LName,_,_,_,_) <- true.
+!get_location(L) <- L = "unknown".
+!get_activity(ANAme) : name(N)&person(N,ID,_,_,_,_,_,_,_,_)& person_activity(ID,A)
     & activity(A,AName,_,_) <- true.
+!get_activity(A) <- A = "unknown".
```

Having all the information, the actual routing of the call, based on this information, is done by the plan triggered by the addition of the `subgoal process_call`. Certain conditions are verified and the action to be executed depends on this conditions.

```
+!process_call(X, DeviceList, Relationship, Location, Activity) : Location ==
     "office" & Relationship == "family" <- ring_mobile.
+!process_call(X, DeviceList, Relationship, Location, Activity) : Location ==
     "office" & Relationship == "colleague" <- ring_fixed_phone.
```

As stated in the policy, if the doctor is in the office and a member of his family calls, then the call should be routed to his mobile phone. If the call is from a colleague, the fixed phone in his office should ring. The default plan, which is applicable when none of the conditions tested are true, states that his mobile phone should take all other calls.

```
+!process_call(X, DeviceList, Relationship, Location, Activity) : true  <-
ring_mobile.
```

In a similar fashion, a set of plans is defined for each agent in order to cover all the policies for the user it represents.

AgentSpeak(L) includes a mechanism that allows agents to communicate, thus sharing plans and consulting each other about the content of their beliefs base. We found that the power of this mechanism can be used to handle conflicting policies.

For illustrative purposes, we use an example with two common features, Originating Call Screening (OCS) and Call Forwarding (CF). These are two classical features in any telephony system. OCS forbids calling numbers on a screening list, while CF forwards incoming calls to another number.

A conflict (feature interaction) occurs if some user *A*, whose OCS screening list includes another user *X*, calls user *B,* who forwards calls to *X* through CF, thus overruling A's policy.

In order to illustrate how this conflict is solved in our simulator, let us assume that Bob has OCS and forbids any calls to Charles and Alice has CF and forwards her call to Charles.

Thus, agent bob will have in its beliefs set a line saying:

```
ocs(charles).
```

Similarly, agent alice will have a belief:

```
call_forward(charles).
```

When Bob tries to call a number, the event that is generated is `dial(X)`, where X is the person that he wants to reach. The event will trigger the execution of a plan. It is checked if the person that is called is on the screening list and only if is not the call is completed.

```
+dial(X) : ocs(X)<- .print("You are forbidden to call ", X).
+dial(X): true<- .print("Inviting ",X); .send(X,tell,incoming_call(bob)).
```

At the other end, when Alice (who forwards all her calls to Charles) receives an incoming call, her agent checks if the call should be forwarded. If this is affirmative, then it will ask the originator of the call if this can be done (it actually asks if the person where the call is about to be forwarded is on the originator's screening list) and upon confirmation, the call will get through.

```
+incoming_call(X) : true <-  ?call_forward(F);
      .send(X,askIf,ocs(F),Answer); .print("answer from a1: ", Answer);
      !ans(Answer,X,F).
+!ans(true,X,Y,F) : true <- .print ("not allowed to connect ", X, " to ", F).
+!ans(false,X,Y,F) : true <- .print("forwarding call from " , X, " to", F);
      .send(F,tell,incoming_call(X)).
```

This simple example shows how the mechanism for agent communication can be used in negotiation of conditions and preferences for users and allows for resolution of conflicts. The introduction of context allows for much richer policies that may handle calls depending on presence, availability, role, capability, call type or call content. Conflicts can occur also between these policies and the built-in mechanism for agent communication in AgentSpeak(L) allows for easy resolution.

This approach has been applied to other features and combinations of features not described here.

7. Related work

In recent years, there has been a lot of interest in service integration, resulting from the increasing interest of users to customize their services. The Universal Inbox project [31] defines an architecture for building personal and service mobility features. The Mobile People Architecture [33] is an architecture based on a Personal Proxy for achieving person-level routing.

Mercury [34] is a system that supports unified communication. It allows a person to initiate a conversation with another party using any available device. The system routes a call with consideration of the device the other party prefers to use in a given context. While Mercury uses SIP as the underlying mechanism for creating, maintaining, and terminating sessions, our goal was to make our architecture protocol independent. The functional entities of our design are independent of the underlying communication mechanisms. Thus, our solutions are compatible with both SIP and H.323.

With respect to Jason and its applications, we can report the work presented in [34]. The authors present a platform for multi-agent based social simulation. The platform is called MAS-SOC, and the approach to building multi-agent based simulations with it involves the use of Jason.

8. Conclusions

We have proposed an agent-oriented architecture for the provision of enhanced services for users of personal communications systems and ubiquitous computing, as well we have focused on the use of AgentSpeak(L) for the double role of implementation and policy language, following important developments in the area of agent-oriented programming and multi-agent systems. Communication and telephony are areas of application where, to the best of our knowledge, the BDI architecture has not been used, yet their complexity requires the sophisticated control that can be readily expressed in AgentSpeak(L). More than other BDI models, AgentSpeak(L) has an exact notation, as well as a clear and precise logical semantics, which resulted in the successful implementation of its abstract interpreter [27]. AgentSpeak(L) enables an elegant specification of the BDI agents and allows to model how agents can search and expand planned actions in order to select alternatives. In addition, it allows inter-agent negotiation for the resolution of interactions.

Another contributions of our work is our concept of “Consolidated Presence Information”[2]. Consolidating all the information that is available for users and their environment provides a unified view of the status of the user at a given time. The architecture allows real-time use of this information, thus simplifying the implementation of a large spectrum of complex services.

Acknowledgment. This work was funded in part by the Natural Sciences and Engineering Research Council of Canada. We are grateful to Prof. Babak Esfandiari and Prof. Ahmed Karmouch for detailed remarks on earlier versions of this manuscript.

REFERENCES

1. Turner, K., Magill, E., Marples, D. – Service Provision. Technologies for next generation communications, Wiley Series in Communications Networking & Distributed Systems, 2004.
2. Plesa, R., Logrippo, L. – Enhanced communication services through context integration, In T. Magedanz, A. Karmouch, S. Pierre, I. Venieris (Eds) - Mobility Aware Technologies and Applications – (MATA, Montreal Oct. 2005) Short papers volume, 1-5.
3. Plesa, R., Logrippo, L. An Agent-Based Architecture for Contex-Aware Communications. Proc. of the 21st International Conference on Advanced Information Networking and Applications Workshops (PCAC-07 Niagara Falls, May 2007), IEEE Press, Vol.2, 133-138.
4. Day, M., Rosenberg, J., Sugano, H. - A Model for Presence and Instant Messaging, IETF RFC 2778, February 2000.
5. Rosenberg, J. SIP Extension for Presence, IETF Internet Draft, 2001
6. Moran, T., Dourish, P. – Context-Aware Computing. Introduction to the special issue on Context-Aware Computing. Human-Computer Interaction, 16 (2-3) 2001
7. Ryan, N., Pascoe, J., Morse, D. – Enhanced Reality Fieldwork. The context-aware archeological assistant, http://www.cs.ukc.ac.uk/projects/mobilcomp/FieldWork/Papers/CAA97/ERFldwk.html, 1997.

8. Schmidt, A., Beigl, M., Gellersen, H. – There is More to Context than Location, Computers and Graphics, vol. 23/6, 893-902, Dec. 1999.

9. Sugano, H., Fujimoto, S. – Presence Information Description format, IETF RFC 3863, 2004

10. Rosenberg, J., Schulzrinne H. - SIP – The Session Initiation Protocol, IETF RFC 2543, 1999

11. H.323 Information Site - http://www.packetizer.com/voip/h323/

12. Wennlund, A. - *Context-Aware Wearable Device for Reconfigurable Application Networks*, Department of Microelectronics and Information Technology (IMIT), Royal Institute of Technology (KTH), Master of Science Thesis at the Royal Institute of Technology (KTH), Stockholm, Sweden, 2003

13. Chen, G., Kotz, D., - *Solar: A pervasive-computing infrastructure for context-aware mobile applications*, Department of Computer Science, Dartmouth College, Hanover, NH, USA, 2002.

14. Amyot, D., - Introduction to the User Requirements Notation: Learning by Example. Computer Networks, 42(3), 285-301, June 2003.

15. Lennox, J., Schulzrinne, H. - Call Processing Language Framework and Requirements, IETF Internet Draft CPL-Framework-02.

16. Wu, X., Schulzrinne, H., - LESS: Language for End System Services in Internet Telephony, IETF Internet Draft, 2005

17. Reiff-Marganiec, S., Turner, K. J. - APPEL: The ACCENT project policy environment/language, Technical Report CSM-161, University of Stirling, UK, 2004

18. Rao A. S., Georgeff, M. P. - BDI-agents: from theory to practice. In Proceedings of the First International Conference on Multiagent Systems, San Francisco (ICMAS-95), 312-319, Dec. 1999,.

19. Wooldridge, M. J. - Reasoning about Rational Agents. Intelligent Robots and Autonomous Agents Series, The MIT Press, 2000.

20. Heinze C., Goss, S., - Human Performance Modelling in a BDI System. In Proceedings of the Australian Computer Human Interaction Conference, 2000.

21. Howden, N., Ronnquist, R., Hodgson, R., Lucas, A. - JACK Intelligent Agents: Summary of an Agent Infrastructure. In Proceedings of the 5th International Conference on Autonomous Agents, 2001.

22. Georgeff, M., Lansky, A. - Procedural Knowledge. In Proceedings of the IEEE Vol. 74 (10), 1383-1398, 1986.

23. Huber, M. - Jam: A BDI-theoretic Mobile Agent Architecture. , In Proceedings of The Third International Conference on Autonomous Agents, 236-243, 1999

24. Rao, A.S., Georgeff, M.: Decision procedures for BDI logics. Journal of Logic and Computation 8**,** 293 – 342, 1998.

25. Winikoff, M. - AgentTalk Home Page. http://goanna.cs.rmit.edu.au/ ~winikoff/agenttalk (2001)

26. d'Inverno, M., Hindriks, K.V., Luck, M. - A formal architecture for the 3APL agent programming language. In Bowen, J.P., Dunne, S., Galloway, A., King, S., eds.: Proc. of the 1st International ZB Conference, Springer Verlag 168–187, 2000.

27. Rao, A.S. - AgentSpeak(L): BDI agents speak out in a logical computable language. In de Velde, W.V., Perram, J.W., eds.: Agents Breaking Away. Springer Verlag 42–55 LNAI 1038, 1996.

28. Jason - http://jason.sourceforge.net

29. d'Inverno, M., Kinny, D., Luck, M., Wooldridge, M. - A formal specification of dMARS. In Singh, M.P., Rao, A., Wooldridge, M., eds.: Proc. of the 4th International ATAL Workshop, Springer Verlag (1997) 155–176 LNAI 1365, 1997.

30. Caire, G., Lhuillier, N., Rimassa, G. - A communication protocol for agents on handheld devices. In Proceedings of the Workshop on Ubiquitous Agents on Embedded, Wearable and Mobile Devices, 2002.

31. Maamar, Z., Mansoor, W., Mahmoud, Q. H. - Software agents to support mobile services. In Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems, 666–667, 2002.

32. Raman, B., Katz, R., Joseph, A. - Universal Inbox: Providing Extensible Personal Mobility and Service Mobility, in Proc.of the 3th Workshop on Mobile Computing Systems and Applications, 2000.

33. Roussopoulos, M. and et.al - Personal-level Routing in the Mobile People Architecture, in Proceedings of the USENIX Symposium on Internet Technologies and Systems, 1999..

34. Lei, H., Ranganathan, A. - Context-Aware Unified Communication, 2004 IEEE International Conference on Mobile Data Management, 176-186, 2004.

35. Bordini, R.H., Rocha Costa, A. C., Hubner, J.F., Moreira, A.F., Okuyama, F.Y., Vieira, R. - MAS-SOC: a Social Simulation Platform Based on Agent-Oriented Programming*,* Journal of Artificial Societies and Social Simulation 8, (3), 2005.

Index