Agent-based Support for Mobile Users using AgentSpeak(L)

Talal Rahwan[1], Tarek Rahwan[2], Iyad Rahwan[3], and Ronald Ashri[1]

[1] School of Electronics and Computer Science, University of Southampton, Southampton, UK
tr03r,ra@ecs.soton.ac.uk
[2] Faculty of Computer Engineering, University of Aleppo, Aleppo, Syria,
tarek_rahwan@hotmail.com
[3] Department of Information Systems, University of Melbourne Melbourne, Australia
i.rahwan@pgrad.unimelb.edu.au

Abstract. This paper describes AbIMA, an agent-based intelligent mobile assistant for supporting users prior to and during the execution of their tasks. The agent is based on the well-known AgentSpeak(L) agent architecture and programming language, which provides explicit representations of agents' beliefs, desires and intentions (BDI). AbIMA is implemented using Java 2 Mobile Edition and is tested on a hand-held computer. We also provide conceptual foundations and discuss various challenges relating to the use of cognitive agent architectures for intelligent mobile user support.

1 Introduction

In recent years, the use of mobile handheld computing devices has been on the increase. They offer a wide variety of services to mobile users, such as information provision and management [22]. However, there are still many opportunities for providing more sophisticated, *intelligent* services to users on the move. Such services should go beyond basic information provisioning and organisation tasks by providing intelligent, pro-active support to mobile users prior and during the execution of their tasks.

Consider an engineer following a project agenda, which involves the completion of a variety of tasks. Suppose that the engineer is using a mobile computing device to help manage the work agenda, conduct meetings with clients and consultants, travelling to a variety of building sites, and so on. In the process of task execution, the engineer may face unexpected situations. For example, a meeting may take longer than expected, or a technical problem on site may preclude the completion of one stage in the building process. A useful intelligent support system must be capable of dealing with such unexpected situations and provide the user with alternative *plans* for achieving the objectives. Such a system must be capable of acquiring information about the environmental context, the objectives required, the alternative means through which such objectives may be achieved, and most importantly, it must be able to reason about all of these concepts in order to provide simple, coherent support to the user. Agent-based approaches have become increasingly popular due to their ability to produce modular

software systems capable of providing intelligent assistance in dynamic, unpredictable environments [11].

An intelligent agent is an agent that is capable of flexible autonomous behaviour [11], where flexible means:

1. *Responsive (or reactive)*: able to perceive the environment and respond in a timely fashion.
2. *Proactive*: exhibit goal-directed behaviour and take initiative when appropriate.
3. *Social*: able to interact with other agents or humans when needed.

Based on this underlying understanding of agent software, agents seem to offer a set of capabilities that are very closely aligned with the requirements of applications for mobile users. This is true for the following reasons. Firstly, a mobile user is usually situated in some environment, which can be represented in terms of context information, such as the time, place, and task at hand. Secondly, the environment is dynamic, since users may move from one place to another, and since their tasks may change based on their circumstances. Following the example, the engineer may move from one site to another, and may have different tasks to achieve, etc. Thirdly, a mobile computer system must have the ability to be proactive, reasoning about the user's goals and how they may be achieved. Finally, in mobile settings, there is a need for the ability to interact with the user, and potentially with other agents. For example, an agent working on behalf of an engineer may interact with agents representing other engineers to sort out meetings and schedule joint tasks.

Some of the most successful theoretical frameworks of rational agents are those belonging to the family of Belief-Desire-Intention (BDI) architectures [5, 17, 18]. It has been argued that these three elements of an agent's mental state can provide a basis of rational action. The agent has explicit representations of its beliefs about itself and its environment, its desires (or goals),[4] which are states of the world it seeks to bring about, and its intentions, which are active plans that the agent adopts in an attempt to achieve these desires. BDI agents have proven useful in the theoretical study of rational agents [21] and in practical applications, for example, in the telecommunications industry [14] and the defence industry [8]. Even though there is a body of research on implementing agents on mobile and handheld devices (e.g., [1, 12]), no attempt, to our knowledge, has been made to implement a specific logic-based BDI architecture on these devices.

This paper represents a comprehensive attempt to implement a Belief-Desire-Intention agent architecture and programming language, specifically the AgentSpeak(L) architecture [17], on a mobile device. By doing this, the project advances the state of the art in two ways: (i) The project investigates how a cognitive model of computational agency, and in particular a BDI framework, may assist in building applications that provide useful, intelligent support for mobile users in realistic settings. (ii) In doing so, the project also attempts to tackle challenges relating to automated support and volatile user behaviour. This is demonstrated through a pilot agent application for supporting a student in completing a number of tasks on campus.

[4] In the remainder of the paper, we shall use the terms 'desire' and 'goal' interchangeably.

This paper, which is an extended version of [16], is organised as follows. In the next section, we present a scenario involving a student attempting to complete a number of tasks on campus. We use this scenario to extract some of the core requirements needed in an intelligent mobile software assistant. In section 3, we give a brief introduction to the AgentSpeak(L) agent programming language, showing how it has been used in a novel way to provide intelligent assistance. In section 4, we demonstrate AbIMA through a simple illustrative example, showing the user interface and screen-shots. Section 5 shows some of the emerging challenges that we have encountered, and which provide fertile ground for future research. We finally discuss related work in section 6, followed by a conclusion in section 7.

2 Motivations and Design Requirements

In this section, we explain in more detail the reasoning behind the adoption of an agent-based approach to support mobile users. In particular, we present a specific scenario in which it would be helpful to provide task support for the user. Subsequently, we outline the essential features that need to be provided by the support system, and explain how a BDI agent could provide some of the required functionality through its interactions with the user.

2.1 Scenario: Student on campus

We begin by outlining a specific scenario through which we can extract the essential requirements for task support for a mobile user in a dynamic environment. Consider a student, named Omar, on his first day at the university who needs to achieve the following major objectives during the day:

A. Go to the Faculty of Computer Engineering building at the University.

B. Attend a lecture for the subject ”Introduction to Computer Science” from 10:00am to 11:30am.

C. Get his new student card from the university registration department.

D. Meet with his friend Ziad on the way to the Grand House Restaurant for lunch. Ziad studies architecture and will finish his lecture at 12:30pm.

Within an agent context the above tasks may be represented as a set of goals that need to be fulfilled in the given sequence. In order to fulfil each goal, Omar needs to execute a sequence of actions (i.e. to execute a plan). There might be a number of plans for achieving the same task. For example, in order to achieve the first goal, there might be two possible plans:

Plan A.1:
A.1.1: Wake up at 09:00am;
A.1.2: Get a taxi;

A.1.2: Ask the taxi to go to the Faculty of Computer Engineering.

Plan A.2:

A.2.1: Wake up at 8:15am;
A.2.2: Get a service bus from home to the main City Square;
A.2.3: Get another service bus from the City Square to the University Square;
A.2.4: Walk up the university road until you find the building.

The plan Omar will actually follow will depend on a number of factors, such as his preference for what time to wake up, his current budget, and so on. As a result, Omar needs to perform *plan selection* based on some preference criteria. Suppose Omar chooses to use plan A.2. This plan now becomes an *intention*. In other words, Omar intends to execute this plan in order to achieve his goal.

Now, in order to achieve the task A.2.1 of waking up at 8:15am, Omar might set up the alarm clock. Then, he needs to get a service bus to the City Square. This might, again, require the execution of another set of sub-tasks. Omar might consider going to the City Square as a *sub-goal* that is part of his intention towards achieving the *super-goal* of going to the Faculty of Engineering building. And in order to achieve this sub-goal, there might be a number of different possible plans to intend:

Plan A.2.2.a:

A.2.2.a.1: Go to Teshreen Street and board service bus number 12.
A.2.2.a.2: Get off at the City Square.

Plan A.2.2.b:

A.2.2.b.1: Go to Nile Street and board service bus number 8.
A.2.2.b.2: Get off at the City Square.

In this manner, Omar can continue, attempting to achieve all of his goals by executing the appropriate plans, which may trigger other sub-goals, and so on.

In the process of executing his plans, however, a number of problems may arise. For example, suppose that after reaching the City Square, Omar is not able to take a bus to the University Square because the bus line is not operational (say because the roads are closed for a diplomatic visitor). In such a case, Omar should be able to find an alternative plan for reaching the university, say, by taking different bus lines. In other words, Omar needs to perform some form of *plain failure recovery*. The new alternative plan must take into account the new context.

Another potential problem Omar might face is *conflict between different goals*. Suppose, for example, that Omar wishes to play football with his friends that morning. Obviously, this goal conflicts with the goal of attending the morning lecture. Omar might be able to *resolve* that conflict by arranging for a different time to play football, or might have to perform *goal selection* in order to make a decision about which goal is more important.

2.2 Requirements for Intelligent Mobile Assistance

It is clear from the scenario we presented in the previous subsection that as the number of tasks and alternative plans involved increases, the complexity of the user's agenda increases significantly. This creates the opportunity for providing automated support for the mobile user. In this paper, we are interested in supporting the user by providing appropriate *advice*, in the form of suggested plans of action, throughout the execution of his/her tasks. Such support must overcome particular challenges arising due to the dynamic nature of the environment and the mobility of the user. The following are the core essential features that a software system providing support to mobile users must provide.

Mobility The entire implementation must be adapted to the capabilities of mobile devices, taking into account their restrictive user interface, limited computational power and reliance on finite power supply.

Practical Reasoning Given a specified set of user goals, the system must be able to perform some form of planning in order to generate a set of actions that, if performed by the user, would achieve these goals.

Context-sensitivity Planning must take into account the current context in which the user is situated (e.g., the current user's physical location or the latest changes in the bus timetable). Information about contextual changes may be provided manually by the user, or be automatically detected (e.g., changes in location could be tracked using a Global Positioning System (GPS) device).

Plan selection If there are a number of alternative plans for achieving the same goal, the system must be able to make a choice based on some comparison of the different plans. This may depend, for example, on the time needed, the overall cost, the risk factor, the user preferences, etc. Appropriate decision mechanisms must therefore be supplied for supporting plan selection.

Plan failure recovery If a plan fails at some stage, the system should be able to retract appropriately and select another alternative plan to suggest to the user. In order to allow for plan failure recovery, the interface must allow the user to easily indicate which parts of the plan have become unachievable and, possibly, for what reasons.

Conflict resolution and goal selection Sometimes, the user might have a number of goals that cannot be achieved simultaneously. In such cases, the system must be able to make a decision about which goals to try to achieve. In making such decisions, the system needs to take into account the importance of the goals as well as the costs of executing the plans.

In order to provide automated user support that has the above features, we propose an agent-based intelligent mobile assistant (AbIMA), running on a hand-held computer (e.g., a Palm Pilot). AbIMA will enable the user to enter information about the user's goals (which represent the high-level tasks, such as attending a lecture) as well as beliefs about the world, the user's preferences, and so on. Then, AbIMA will reason about these goals, in the context of the beliefs it has acquired from the user and the environment, and provide the user with the appropriate plan of action.

In seeking a computational architecture that is suitable for providing the required functionalities described above, we found a significant overlap between the requirements of our domain and the concepts discussed in the belief-desire-intention (BDI)

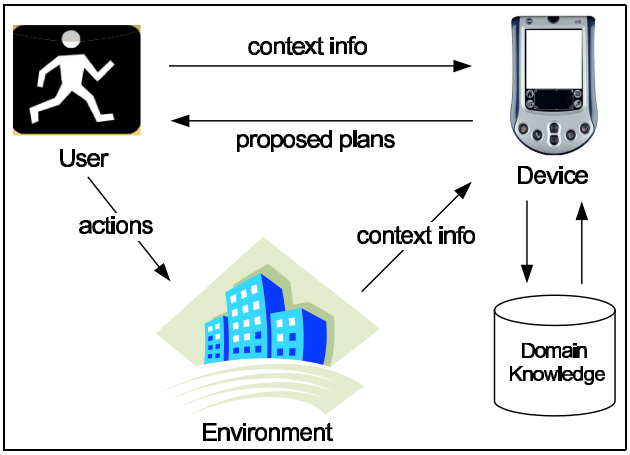

Fig. 1. Mobile device support for mobile users

view of agency [3, 5, 18]. In particular, BDI-style agent architectures require explicit representations of agents' mental attitudes, such as beliefs, desires (or goals), plans, intentions (i.e. active plans), and so on. Various BDI-style architectures have provided different ways in which agents may reason about their mental attitudes in dynamic environments. Hence, BDI agent architectures seem to offer a promising basis for providing the features required in supporting mobile users.[5]

In the context of our application, AbIMA will exploit the rich representation of BDI agents to capture information about Omar's beliefs, constraints, goals, preferences, plans, and so on. Then, AbIMA will use agent-based techniques to reason about these mental attitudes and make decisions about what plans (and subsequently, what actions) to suggest to the user. Another attractive notion of BDI architectures is that they are usually based on the aggregation of plans from plan libraries, encoded for a particular domain. This results in significant reduction of the complexities encountered in traditional planning systems, which is particularly helpful within the context of operation on limited mobile devices. Hence AbIMA will have a repository of pre-programmed plans that can be used and combined in order to achieve different goals. In particular, AbIMA is based on a specific BDI architecture and programming language called AgentSpeak(L) [17],[6] which we completely re-implement for running on mobile devices. In the subsequent section we discuss in more details the rationale behind the choice of AgentSpeak(L), as opposed to alternative BDI architectures.

[5] We concede that popular BDI architectures, and AgentSpeak(L) in particular, do not provide the details of *all* the features we need (such as mechanisms for reasoning about constraints and about conflicts between plans). Nonetheless, the BDI view offers a useful framework within which one may design these features and integrate them in a coherent manner.

[6] AgentSpeak(L) can be seen both as an agent architecture and an agent programming language. It is an architecture, in the sense that it has multiple interacting computational components, and a language, in the sense that it is executable from declarative program specifications.

2.3 Interaction between user, environment, and device

The interactions between user, device and the environment are illustrated in Figure 1. The device provides the user with suggested plans. In order to do so, the device uses its general domain knowledge, as well as dynamic contextual information about the environment and the user. The user then acts in the environment (e.g., moves from one place to another, or completes a school assignment). These changes are observed by the device agent (either automatically or through user input), which might then make appropriate changes to the user's agenda.

Based on these interactions, the user and intelligent agent program can be viewed as offering each other complementary services based on their individual capabilities. The user and agent cooperate, each one of them dealing with those aspects of the problem they are best suited to. On the one hand, the agent has access to detailed, in-depth information about task plans, conflicting engagements, and rules for dealing with conflict. On the other hand, the user provides the necessary contextual information based on the current environmental situation, his changing goals, and so on.

3 AgentSpeak(L) for Intelligent Assistance

In the previous section, we outlined the core requirements needed in AbIMA and briefly outlined why an automated assistant agent based on a BDI architecture has the potential to fulfill these requirements. In this section, we describe the AgentSpeak(L) programming language in some detail and show how some of the features needed in AbIMA may be implemented in AgentSpeak(L).

AgentSpeak(L) is an agent architecture/language with explicit representations of beliefs, goals, and intentions. It was initially introduced by Rao [17], who provided a language for specifying BDI agents with a sketch of an abstract interpreter for programs written in the language. This, it was argued, provided the first bridge of the gap between BDI theory and practice. One reason we have chosen AgentSpeak(L), as opposed to other BDI models, is because it has an exact notation, thus providing an elegant specification of BDI agents. Moreover, AgentSpeak(L) has a clear, precise logical semantics, as well as being described in a computationally viable way [7]. This resulted in successful implementation of its abstract interpreter (as in [13], for example).

We now present a very brief overview of the AgentSpeak(L) syntax and its informal semantics. An AgentSpeak(L) agent is created by specifying a set of beliefs (called the *belief base*) and a set of plans (called the *plan library*). A *belief atom* is a predicate (as in Prolog). A *belief literal* is an atom or its negation. There are two types of goals in AgentSpeak(L). An *achievement goal* is a predicate prefixed with "!", stating that the agent wants to achieve a state of the world where the predicate is true. A *test goal*, on the other hand, is a predicate prefixed with "?" and states that the agent wants to test whether the associated predicate is a true belief (i.e., whether it can be matched with the agent's belief base).

A *triggering event* in AgentSpeak(L) is an event that causes a plan to be executed. There are two types of triggering events: events involving the addition "+" of a mental

attitude (e.g., a belief or a goal), and events involving the deletion "-" of a mental attitude. Let us now explain how plans are represented [17, definition 5].

Definition 1. *(Plan) If e is a triggering event, $b_1, ..., b_m$ are belief literals, and $h_1, ..., h_n$ are goals or actions, then a plan is represented as follows*

$$e : b_1, \ldots, b_n \leftarrow h_1; \ldots; h_n$$

The expression on the left of the arrow is the head *of the plan, and the expression to the right of the arrow is the* body *of the plan. The expression $b1, , bm$ is referred to as the context in which the plan becomes* applicable.

Consider the AgentSpeak(L) plan below, expressing plan A.2 from section 2.1 above:[7]

Plan A.2
$+!location(user, faculty, 10am)$: $location(user, home, 8\_15am)$
$\leftarrow wakeup(user); !bus(home, city\_square);$
$!bus(city\_square, uni\_square); walk(uni\_square, uni\_rd)$

This plan states that in the event of adding a goal that the user be at the faculty at 10:00am, if the current context condition states that the user is at home and the current time is 8:15am, then the goal may be achieved by waking the user, taking a bus from to the City Square, taking another bus from the City Square to the University Square, and finally walking from the University Square up the University Road. Recall that elements of the plan body may be directly executable actions (e.g., walking up the University Road), or may be sub-goals which themselves need plans to be achieved (e.g., taking a bus to the City Square involves going to the bus stop, boarding a particular bus, etc.)

Note that in AgentSpeak(L), an event may be external or internal. An external event is one that is originated by a perception of the environment or from a given basic goal. In the above example plan, the goal $+!location(user, faculty, 10am)$ is an external event, since it is a goal provided by the user. An internal event, on the other hand, is one that is generated during the agent's process of plan execution. For example, in the course of executing the plan above, the second sub-goal of taking a bus from home to the City Square is fired as an internal event. This event may then trigger the plan described below , which corresponds to plan A.2.2.a described in section 2.1.

Plan A.2.2.a
$+!bus(home, city\_square)$: $location(user, home)$
$\leftarrow walk(home, teshreen\_rd); take\_bus\_number(12);$
$get\_off(city\_square)$

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 referred

[7] Note that actual plans can be more generic, using variables for time, locations, etc. and retrieving those from knowledge sources such as local or network databases. For simplicity of presentation, however, we use instantiated plans.

to as an intention associated with that goal. An AgentSpeak(L) agent can be described formally as follows [17, definition 6]:

Definition 2. *(Agent) An agent is a tuple*

$$\langle E, B, P, I, A, S_E, S_O, S_I \rangle$$

where E is a set of events, B is a set of base beliefs, P is a set of plans, I is a set of intentions, and A is a set of actions. The selection function S_E selects an event from the set E; the function S_O selects and option or applicable plan from a set of applicable plans; and S_I selects an intention from the set I.

We now explain the basics of the AgentSpeak(L) interpreter. At every cycle, AgentSpeak(L) updates the list of events. This causes relevant addition or removal of goals as well as possible updates to the belief base B. Updating beliefs must be done appropriately in order to ensure that B remains consistent. This may be done using some appropriate Belief Revision Function (BRF) [13]. The selection function S_E now selects an event from the list E. This event is unified against the triggering events at the heads of plans, generating the set of relevant plans. Now, for each plan, the context is tested against the agent's beliefs. Those relevant plans that are unified successfully are the applicable plans. The function S_O now selects one of these plans. If the event was external, a new intention is created and placed in I. If, on the other hand, the event was an internal one, the selected plan is placed on top of the plan stack of an existing intention. Now, the function S_I selects an intention for execution. Recall that executing an intention may involve either executing direct actions, or may involve triggering sub-goal events to be triggered. If the sub-goal is an achievement goal, it causes an internal event to fire; this may subsequently cause new plans to be added to the stack of plans associated with that intention (this happens in future reasoning cycles). If, on the other hand, the sub-goal is a test goal, the belief base is consulted to test whether the associated predicate is true. Actions and goals that are achieved are removed appropriately from the intention stacks. This ends a cycle of the interpreter. AgentSpeak(L) starts over again by checking the state of the environment and updating the belief base accordingly, generating events, and so on.

We now go through how some of the stages of the interpreter may be executed in our applications. In the context of an agent supporting a mobile user, an external event may be one of the following:

1. *User inputs goal.* At any stage, the user inputs the high-level goals he wishes to achieve. In our example, these would be represented by the tasks A, B, C and D described in the scenario above. These cause external achievement goal events of the form $+!goal(parameters)$ to be added to the set E. The user may also add test goals in the form of queries (e.g., the user may enquire about the current time, the location of a lecture theatre, and so on).
2. *User removes goal.* The user must also be allowed to remove goals, causing events of the form $-!goal(parameters)$ to be fired.
3. *User inputs or modifies belief.* The agent may perceive some new information about, or changes in the environment. For example, the user may perceive that the buses from Teshreen Road to the City Square have been cancelled. In this

case, the user may notify the agent of these changes by adding events of the form $+belief\_predicate(parameters)$ and $-belief\_predicate(parameters)$. Instead, the agent might be able to automatically retrieve such information, say from web services for bus timetables, or location positioning systems.

AbIMA may retrieve beliefs about Omar's current location, which is home, the current time, and so on. Omar would also inform the agent of its goal to reach the Faculty at 10:00am on a particular day by adding $+!location(user, faculty, 10am)$. After being selected by the event selection function S_E, this goal may be unified with the trigger event in the plan A.2. When the context condition of that plan is satisfied (i.e., when, based on the agent's clock and its beliefs about the users location, predicate $location(user, home, 815am)$ is true in the belief base), the plan will become applicable. This plan will be compared with other possible applicable plans. If there are no other applicable plans at this stage, or if this plan is the most preferred, the selection function S_O will select this plan.[8] Since we are still at the top-level goal, a new intention will be generated and the plan will be added on top of the stack associated with that intention.

Now, the agent begins executing the plan. This involves informing the user about what actions he needs to take in order to achieve his goal. The action represented by the predicate $wakeup(user)$ is an atomic action and may be executed directly by activating the alarm on the Omar's handheld device. On the other hand, the activity of taking a bus to the City Square, represented in the predicate $bus(home, city_squ are)$, is not an atomic one; it is an achievement sub-goal that needs another plan to be executed before we can say it is achieved. This goal is posted as an internal goal, updating the set E of events. This event may then follow a similar process, triggering the plan A.2.2.a, and so on, until all activities in the plans reach the atomic level and hence can be executed directly, either by the agent or by Omar.

Note that during the agent lifecycle, unexpected events may also occur, such as a plan failing to be achieved due to the user's failure to achieve some basic action. For example, Omar may fail to take a particular bus due to road works. This requires a plan failure operator, which involves removing certain intentions, removing sub-goals, notifying the user, and so on. In this version, we do not fully deal with these issues.[9]

4 Demonstrating AbIMA

In this section, we demonstrate the AbIMA user interface and explain how it supports the user through part of the scenario described in section 2.

We have implemented a complete version of the AgentSpeak(L) agent programming language on a Palm Pilot handheld device, running Palm OS, using Java 2 Micro Edition [10]. This forms the underlying engine of AbIMA.

[8] This means that the user's preferences over alternative plans need to be encoded within the option selection function S_O itself.

[9] Note that exactly how failure is dealt with was not fully articulated in Rao's original description of AgentSpeak(L) [17]. Therefore, we need to add a plan failure operator that is appropriate for our domain (this could be, for example, based on AgentSpeak(XL) [2]).

Fig. 2. AbIMA introductory screen

Figure 2 shows the introductory screen of AbIMA. The user may choose to view and/or edit the beliefs, desires or intentions stored by AbIMA. In order to illustrate the current functionality of AbIMA, we step through a simple example, which is based on, but slightly different to, the example presented in section 2.

Fig. 3. Attend database lecture plan

Fig. 4. AbIMA goals lists

By clicking on 'intentions', the user may view the current plans suggested by the agent. Suppose, in this case that the agent has generated a plan for attending a lecture on 'databases' at 9:30am. The result is shown in figure 3.[10] The first action in the plan involves waking up at 8:15am. This is executed by the agent by activating the alarm of the PDA; the agent receives confirmation that this action has been executed after the

[10] Currently, AbIMA presents suggested plans as stand-alone lists. One could, however, integrate this list into an existing calendar application.

user sets the alarm off (assuming the user is responsible and does not just go to sleep). Then, the agent advises the user to walk to the Nile Street, take the Nile bus, get off at the City Square, take the Muhandicine bus, get off at the 'Mech' bus stop, and finally walk to the Faculty of IT. It is possible to include 'user interface plans' based on the current context. For example, when the user reaches the City Square, the agent may display a map of the area indicating the location of the Muhandicine bus stop.

Fig. 5. Buy CD plan

Fig. 6. Plan incorporating both user goals

By clicking on 'desires', the user may add a new goal. Suppose the user would like to add a goal of getting a CD from the CD Shop. This means there are now two goals (see Figure 4). In an initial version of AbIMA, the agent generates a plan for getting the CDs given the current context (i.e., given the user is still at home). This results in the plan in figure 5. However, note that this plan also involves going to the City Square. In this case, there is an opportunity for the agent to combine the two plans so as to achieve both goals. This would involve advising the user to pass from the CD shop, while on the way to the lecture. In order to enable AbIMA to perform such reasoning we introduce the notion of *expected beliefs*. These *expected beliefs* are the beliefs that the agent would *expect* to hold true once the second plan (Buy CD) has been achieved. Given these beliefs the agent attempt to find a plan for the first goal (Attend database lecture) from this new context. If a plan can be found that corresponds to a state that the agent would find itself in any case (i.e. beginning from the City Square) the two plans are merged to produce a common plan for both goals. This allows AbIMA to recognise the opportunity to get the CD while in the city. AbIMA hence generates the plan shown in figure 6.[11] Such a solution is certainly not general and can only work under the limited context of common locations and time restrictions, but proves effective in a wide variety of settings, which indicate that it represents a worthwhile avenue for further investigation.

[11] We shall provide the details of this solution in a future paper.

5 Implementation challenges

During our work on AbIMA, we have identified a number of important challenges. While the use of the AgentSpeak(L) architecture and our implementation of it goes some way towards addressing some of the issues, others have been dealt with in an ad hoc manner. Nevertheless, each one of them provides fertile ground for further research. We discuss the most salient of these challenges below.

Limited storage Handheld devices often have limited storage capacity. The agent might need comprehensive timetable information, maps to direct the user, images identifying landmarks to assist the user in establishing his current position, and so on. This creates the need for allowing information modules to be plugged into and out of the device. This could be done through synchronisation with the PC or using a wireless connection. The agent may download these information modules on demand based on the upcoming tasks.

Limited processing power Similarly, due to the limited processing power of mobile devices, special care needs to be taken to ensure proper execution. For example, appropriate design needs to ensure that no extremely long (or infinite) intention stacks are accumulated. Also, the user might need timely response in certain circumstances. This could be facilitated, for example, by producing suggested actions to the user before the complete agent cycle is completed. To avoid these issues, we currently use relatively small plan libraries, belief bases, etc.

Plan configurability and interoperability Due to the limited storage and computational power of handheld devices, the agent architecture should allow for configurable plan libraries that allow the user to supply the agent with the right plans to deal with the upcoming tasks. Moreover, users may wish to download new plans or share plans with other users. One possible solution is to express plans in a shared format based on, for example, the extensible markup language XML.

Online/offline state preservation When users are mobile while engaged in their tasks, they cannot be expected to have the mobile device running all the time. They might also use different devices. This raises the problem of preserving the agent's state properly on the device. This state may also be represented in a modular fashion in order to be backed-up, transported unto a different machine (e.g., desktop computer), and so on.

Perception/action representation Initially, AgentSpeak(L) was designed with a view to implement agents acting autonomously in an environment, such as a robot in a factory cell. In the case of supporting mobile users, on the other hand, additional considerations, relating to the way the agent perceives the environment and acts upon it, must be catered for. For example, we cannot expect the user to have the handheld computer switched on continuously; hence there is no continuous flow of action and perception. Similarly, the limited input capabilities of mobile devices and the limited availability of mobile users (e.g., due to their engagement in the task) may impede the agent's ability to have up to date, detailed information about the environment.

Situation awareness An important challenge related to supporting mobile users is that of modelling situation awareness. How can the agent establish an up-to-date per-

ception of its environment? To address this problem, we have data structures explicitly representing time, tasks, goals, locations, etc., and we allow the user to view and manipulate these structures directly. These data structures can also be updated based on other sources than the user, such as bus timetable databases, etc. Other relevant questions include: If multiple users were connected, how could they establish joint awareness of their corresponding situations? If the user is allowed to switch between different platforms, how is situation awareness affected [19]? These issues are outside the scope of our current work.

As mentioned above, the challenges related to the limited storage and computational capacity of handheld devices may be dealt with by providing a modular, configurable implementation of the AgentSpeak(L) architecture. Therefore, we took a component-based approach to our design with a clear separation between information modules and decision-making modules so that they can be easily changed.

6 Related Work

One related project is the Electric Elves [4], in which multiple agents support each human (e.g., the mobile device, fax machines and desktop computer, each has its own agent) and agents interact to assist people coordinate their activities. The Electric Elves project builds on an ad hoc agent architecture that integrates different artificial intelligence technologies for knowledge representation, planning, teamwork, etc. Our work, on the other hand, focuses on the use of a particular agent programming language and investigates its applicability in the domain of mobile user support. We hope that using a formal agent language such as AgentSpeak(L) would later allow us to formally verify some of the properties of mobile assistant agents, such as adjustable autonomy (where decision making control shifts between the user and the agent), failure recovery, and so on. In this sense, our work is complementary to the Electric Elves project since it enables studying the issue of control flow within a well-defined agent language context.

Another related body of work is the NurseBot project [15]. This project is concerned with the construction of a robot capable of providing personal assistance to the elderly. The robot is capable of performing tasks such as reminding the human of things to be done, helping the human move from one room to another, manipulating objects (e.g., fridge, laundry), and so on. While the application domain of the NursesBot project is different from AbIMA, they share many challenges. For example, both systems require capabilities of planning and reasoning about context. On the technical level, AbIMA differs from NurseBot in that whereas we are proposing to use the BDI architecture as a foundation, NurseBot uses techniques from AI planning and Bayesian reasoning.

Finally, recall that AbIMA needs to plan ahead while making sure plans do not change the context in such a way as to make future plans inapplicable. Some work on detecting and resolving conflicts between plans in BDI agents is presented by Thangarajah et. al. [20]. To achieve the same end, the AgentSpeak(XL) programming language [2] integrates AgentSpeak(L) with the TAEMS scheduler [6]. Moreover, related theoretical work on selecting new plans in the context of existing plans is presented in [9]. We are currently investigating whether such work could be effectively implemented in our domain, given the limitations of mobile devices.

7 Conclusion

In this paper, we presented AbIMA, an agent-based intelligent mobile assistant, running on a hand-held device, for supporting users prior to and during the execution of their tasks. We presented a scenario involving a student executing a number of tasks on campus. We used this scenario to extract the essential features required in an automated agent assisting the user during task execution. We then argued that the belief, desire, intention model of agency seems to cater for these features. AbIMA's implementation is based on the well-known agent architecture and programming language AgentSpeak(L), which provides explicit representations of agents' beliefs, desires and intentions, as well as an abstract interpreter for reasoning about these mental attitudes to guide the performance of actions. We described how the features required in AbIMA can be specified in AgentSpeak(L). Finally, we outlined some technical challenges we faced, hence motivating future work in the area.

While AgentSpeak(L) seems to provide the necessary features required for our application requirements, a number of domain specific issues arise. The agent's perception and action are done through a user engaged in a task. A required action could be performed directly by the device (e.g., activating the wake-up alarm), or indirectly by instructing the user to perform some activity (e.g., to take a particular service bus). The two types of actions must be clearly differentiated and a comprehensive study of the control flow between the user and agent are needed.

Another future direction would be connecting the device into a network of mobile assistants. This raises at least two families of issues. First, there are issues related to teamwork, coordination and negotiation in multi-agent systems. Mobile assistants belonging to different users may interact, for example, to coordinate a meeting, or negotiate a restaurant for a group dinner that satisfies the users' dietary requirements. Second, there are technical issues related to, for example, representation and retrieval of planning information and other information from different sources.

8 Acknowledgements

The authors are very grateful to Rafael Bordini and the anonymous reviewers for their very valuable comments that helped improve this version of the paper. During this work, Iyad Rahwan was supported by a Melbourne University Research Scholarship, and a CMIS–CSIRO Ph.D. Top-Up Scholarship.

References

1. F. Bergenti, A. Poggi, B. Burg, and G. Claire. Deploying FIPA-Compliant Systems on Handheld Devices. *IEEE Internet Computing*, 5(4):20–25, 2001.
2. R. H. Bordini, A. L. C. Bazzan, R. O. Jannone, D. M. Basso, R. M. Vicari, and V. R. Lesser. AgentSpeak(XL): Effi cient intention selection in BDI agents via decision-theoretic task scheduling. In C. Castelfranchi and W. L. Johnson, editors, *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002)*, pages 1294–1302, New York, USA, 2002. ACM Press.

3. M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge MA, USA, 1987.
4. H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. V. Pynadath, T. A. Russ, and M. Tambe. Electric elves: Applying agent technology to support human organizations. In H. Hirsh and S. Chien, editors, *Proceedings of the 13th International Conference of Innovative Application of Artificial Intelligence (IAAI-2001)*. AAAI Press, 2001.
5. P. Cohen and H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
6. K. S. Decker and V. R. Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 2(4):215–234, 1993.
7. M. d'Inverno and M. Luck. Engineering AgentSpeak(L): A Formal Computational Model. *Journal of Logic and Computation*, 8(3):233–260, 1998.
8. C. Heinze and S. Goss. Human performance modelling in a BDI system. In *Proceedings of the Australian Computer Human Interaction Conference*, 2000.
9. J. F. Horty and M. E. Pollack. Evaluating new options in the context of existing plans. *Artificial Intelligence*, 127(2):199–220, 2001.
10. Java 2 Micro Edition. http://java.sun.com/j2me/.
11. N. R. Jennings and M. J. Wooldridge. Applications of intelligent agents. In N. R. Jennings and M. J. Wooldridge, editors, *Agent Technology: Foundations, Applications, and Markets*, pages 3–28. Springer-Verlag, Heidelberg, Germany, 1998.
12. Z. Maamar, W. Mansoor, and Q. H. Mahmoud. Software agents to support mobile services. In C. Castelfranchi and W. L. Johnson, editors, *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002)*, pages 666–667, New York, USA, 2002. ACM Press.
13. R. Machado and R. H. Bordini. Running AgentSpeak(L) agents on SIM_AGENT. In J.-J. Meyer and M. Tambe, editors, *Pre-proceedings of the 8th International Workshop on Agent Theories, Architectures and Languages*, volume 2333 of *Lecture Notes in Computer Science*, Berlin, Germany, 2002. Springer Verlag.
14. H. Nwana, D. Ndumu, D. Lee, and J. Collis. ZEUS: A toolkit for building distribtued multi-agent systems. *Applied Artificial Intelligence*, 13(1):129–186, 1999.
15. M. E. Pollack. Planning technology for intelligent cognitive orthotics. In *Proceedings of the 6th International Conference on AI Planning and Scheduling*, 2002.
16. T. Rahwan, T. Rahwan, I. Rahwan, and R. Ashri. Towards a mobile intelligent assistant: Agentspeak(l) agents on mobile devices. In P. Giorgini and M. Winikoff, editors, *Proceedings of the 5th International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS)*, 2003.
17. A. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. V. de Velde and J. W. Perram, editors, *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume 1038 of *LNAI*. Springer, 1996.
18. A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First International Conference on Multiagent Systems*, San Francisco, USA, 1995.
19. J. C. Tang, N. Yankelovich, J. Begole, M. V. Kleek, F. Li, and J. Bhalodia. ConNexus to awarenex: extending awareness to mobile users. In *Proceedings of the SIGCHI conference on Human Factos in Computing Systems*, pages 221–228, 2001.
20. J. Thangarajah, L. Padhgam, and M. Winikoff. Detecting and avoiding interference between goals in intelligent agents. In G. Gottlob and T. Walsh, editors, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2003)*. Academic Press, 2003.
21. M. J. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.
22. A. Zaslavsky and Z. Tari. Mobile computing: Overview and current status. *Australian Computer Journal*, 30(2):42–52, 1998.