

# Augmenting BDI with relevance

## Supporting agent-based, pervasive applications

Andrew Koster University of  
Utrecht Intelligent Systems  
Group  
koster.andrew@gmail.com

Frank Dignum University of  
Utrecht Intelligent Systems  
Group  
dignum@cs.uu.nl

Fernando Koch University  
of Utrecht Intelligent Systems  
Group  
fkoch@acm.org

Liz Sonenberg University of  
Melbourne Department of  
Information Systems  
l.sonenberg@unimelb.edu.au

### ABSTRACT

The potential of pervasive mobile systems lies in the integration of applications and services available on mobile devices. These services must be built on the premises of context-awareness, such that the application takes advantage of environmental information to improve its performance. The challenge is to process more environmental information with the limited computational resources available on mobile devices. Intelligent Agents have been proposed as a suitable architecture for these tasks. Such agents are designed around a deliberation cycle. We show that one of the problems agents encounter in highly dynamic environments is that this cycle misses opportunities to react to the changing environment. For this reason we propose an extra module that augments agent architectures with a “context-observer”. Based on the structures used in the deliberation cycle, this module evaluates the relevance of received data. It does this parallel to the normal cycle. We illustrate how such an approach could improve the context sensitivity of the agent’s behaviour and thus the usefulness of the application.

### 1. INTRODUCTION

In the past few years there has been a continuing trend towards the use of mobile (wearable) computers in everyday life [6]. One of the largest differences between mobile devices and traditional systems is, of course, their mobility. The user does not need to sit down to use them, but can do so while on the move. For pervasive applications to be effective mediators between the available information and the physical user, they need to present information in a quick and easy manner. As more and more information becomes available, contemporary software has a hard time keeping up. Current mobile devices incorporate many different tools previously found separately, such as a mobile phone, organizer functionalities, integrated sensors, multimedia applications, etc. The true potential of pervasive systems comes from integration of such data [14]. However, current technologies are unfit to handle the multitude and degree of dynamism of mobile and pervasive environmental information. We intend to explore “what” structures in software application are required to process more information (from the environment) with less resources. We are interested in a special kind of software applications: the ones built upon agent technologies. We suggest that agents provide a set of features closely

aligned to the requirements of pervasive mobile applications.

#### 1.1 Using agent technology

In the field of Artificial Intelligence the Intelligent Agent [15] has been proposed as a design method for dealing with issues in robotics, eMarkets and also mobile applications, such as tourist guides, conference assistants [10, 9] and various other task-based applications [4]. Work is being done to extend this to more generally applicable tasks on mobile devices [11]. All these problems have in common that:

- the problem should be solved without requiring the intervention of the user.
- the agent interacts directly with the environment, or suggests such interactions to its user, thus changing its environment.
- the solution requires flexibility. The agent must exhibit proactive goal-oriented behaviour as well as reacting to unforeseen situations.

The unique problem encountered in smartphones, however, is the sheer amount of highly dynamic information in the environment, combined with the limited processing power of a telephone. We wonder *how* this impacts the working of agents.

Many agent architectures are based on the so-called Belief, Desire, Intention (BDI) - Model. This model is based on a respectable philosophical model of human practical reasoning, originally developed by Michael Bratman [2]. It presumes rational entities have desires. To fulfill these desires they form intentions, based on their beliefs about the world. Computational architectures based on this model all work around a deliberation cycle. While there are various different proposals for implementing such a cycle [12, 1], they are based on a similar system:

- **sense** the environment and update the agent’s *beliefs*
- **deliberate** on the agent’s *desires* and plans to achieve them. Form *intentions* and, if things have changed in the environment, possibly reprocess former intentions.
- **act** on whatever intentions the agent decided to execute. Then loop back to sensing.

This cycle is repeated during the runtime of the agent, but how often depends on the architectures. The main processing investment is in the “deliberate” part of this loop and the longer this takes, the larger the chance of discrepancies between the “sensed” environment and the environment that is “acted” in. In a highly dynamic environment it is therefore dangerous to deliberate for too long. A balance has to be found between two extremes. On the one hand *proactively* planning the optimal way to achieve a goal runs the risk that the situation has changed too much to execute it, while on the other hand *reacting* to sensory data with the first plan available may accomplish some action, but it runs the risk that the action is suboptimal or even wrong.

The middle way, and one of the main contributors to success of the BDI-architectures, is splitting the goals up into subgoals that are easier to plan for. The assumption is then that achieving the subgoals moves the agent nearer to achieving the final goal.

The agent executes its immediate actions counting on incomplete information, but the plans may still lead to dead-ends and exceptions. As detailed below, once facing the unexpected, the agent must behave with coherence, by revising what went wrong, and backtracking to the previous selection point. Nonetheless, backtracking is not always possible with physical actions. That is, some actions that changed the situation of the world cannot be undone. Therefore, although ideally we want to plan with the maximum amount of information possible, sometimes the agent has to assume incomplete plans and be prepared to react to unexpected events. This mechanism promotes balance between proactiveness and reactivity [3]. The planning process can think ahead a number of interactions by analysing the tail part of the plan. At the same time, it can reconsider the current basic actions, the remaining sub-goals, or the plan execution itself in reaction to changes in the environment. This requires that the inference system contains the structures to observe the environment and revise the plan execution process. Current BDI-model implementations provide these structures in different architectures.

A BDI-architecture uses the cyclic structure to continuously evaluate the world and update the agent’s actions. An important question is *when* to reconsider its plans and actions? Architectures such as JACK [8] and 3APL [7] detect the failure of a plan once it ends without the (sub-)goal being achieved. They must then process where the plan failed to be able to backtrack. This is an expensive process. We argue that the sooner it is detected that a plan can no longer execute correctly, the easier it is to correct it. However, we also do not wish to ‘baby sit’ the plans, monitoring their progression every step of the way. Instead we propose an extension to the model. In the next section we will show how a simple plug-in module could help in deciding when to start and stop some of the critical processes in the deliberation cycle. In section 2.1 we demonstrate its working with a simple example and finally present our conclusions and discuss future work.

## 2. THE CONTEXT-OBSERVER

When information is liable to change and affect plans at any moment it is unfortunate that it will only be “sensed” in one step of the deliberation cycle. The world is considered stable during the “deliberate” and “act” steps and any information that changes during these stages is only considered

in the next cycle. While speeding up the cycle would allow for more accurate information, it allows for less proactive reasoning. Wouldn’t it be better if sensing the environment is done continuously and separately from the deliberation cycle? The reason for it being in the cycle, however is for the consistency during the “deliberate” step: it creates problems in the deliberation if, while reasoning about a plan, the beliefs it is based on change.

We therefore propose a separate meta-model that preprocesses incoming information. This *context-observer* observes changes in the environment, rationalises the changes that are *relevant* for current deliberations and sends events signaling the occurrences to the planning module. This structure helps to improve deliberation performance by filtering relevant events and promoting real-time coordination between observation and plan deliberation. The intra-module coordination is based on an event passing mechanism.

The deliberation mechanism must then be extended with the ability for events to affect its working by interrupting or starting certain processes. Contemporary agent programming languages such as JACK and 2APL [5] already support this.

### *The way it works.*

We propose the following augmentations to standard BDI-theory [13] to support the evaluation of relevance as described above.

**An agent** is defined as the tuple  $A = \langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \Gamma, \mathcal{A}, \mathcal{R}, \xi \rangle$ .

Where  $\mathcal{B}$  is the beliefbase,  $\mathcal{D}$  is the desirebase,  $\mathcal{I}$  is the intentionbase,  $\Gamma$  is the set of plans,  $\mathcal{R}$  is the set of *relevance filters* and  $\xi$  is the eventbase.

In addition an agent has basic actions  $\mathcal{A}$ .

- *Beliefs* represent the state of the environment, such as items, people, obstacles as well as the robot’s own state, such as current location.
- *Desires* give the overall goals of the agent; that is generic goals the robot should pursue overtime when the proper conditions arrive. If the condition is true the desire is adopted.
- *Intentions* represent the goals to which the agent is committed. These are always preceded by a condition. This describes the necessary preconditions to achieve the intention.
- *Plans* are practical reasoning rules defining *how* the agent infers what to do to achieve a goal. A plan may contain a set of sub-plans (or sub-goals) that would lead to achieving the main goal. Plans are also always preceded by a condition. This describes the necessary preconditions to execute the plan.

These structures are similar to those already in use in agent models, but we have added ‘conditions’ for desires and intentions, rather than only have them for plans. Such conditions should be interpreted as a subset of possible predicates in the world. A condition is considered to be true if the beliefbase is consistent with it. In addition to these variations on standard structures, we introduce events and relevancy filters as new structures for the context-observer.

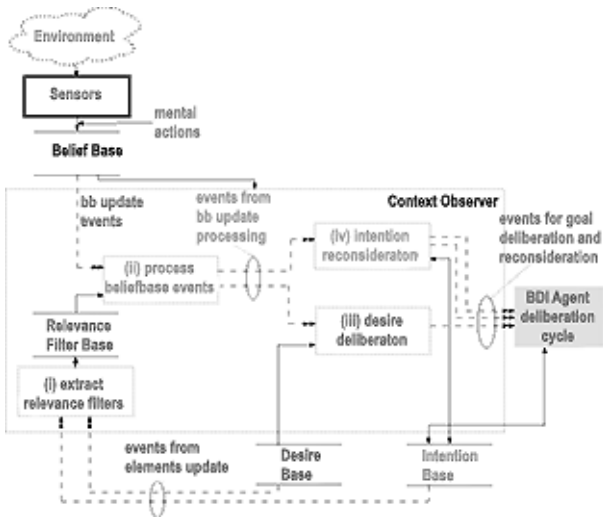


Figure 1: Context-observer design

### Events, relevancy filters and revised deliberation.

Events are used for internal communication in the agent. They keep the other structures and modules updated and the deliberation cycle streamlined. Their structure consists of a message, a triggering object and a receiver.

Relevancy filters are the “planning rules” of the context-observer: they are generated from the conditions of the classic elements and “trigger” if any incoming information changes the truth value of the condition (or conditions).

The context-observer’s ‘deliberation’ will run in parallel to the agent’s deliberation, allowing it to use the resources available as efficiently as possible by preprocessing incoming data based on the *relevancy filters*. If a relevancy filter triggers that a condition, previously true, has been falsified, it triggers reconsideration of the corresponding element.

When the desires, intentions or plans of the agent are changed, then an event is sent to the context-observer. In response to this the context-observer processes the event and updates the relevancy filters. If the element was asserted, all the predicates in the condition are added as a relevancy filter. If the element was retracted, then any predicates not in use for any other conditions are removed from the relevancy filters.

An overview of how all these elements come together can be seen in figure 1. We will now demonstrate how this improves the functioning of the agent with an example.

## 2.1 A context-observer in Tileworld

Let us assume the problem scenario in figure 2. This scenario is a mix between the concepts of mobile services and “Tileworld” games. *The agent has the desire  $goBase(G)$  to reach base  $G$ ,  $downloadFiles(G)$  to download the files for goal  $G$ , and the plan rules to resolve these goals. New bases are added to the scenario at run-time, with different priorities. The number of files to download and the size of the window of opportunity (WoO) varies randomly per goal. Let us say that for each base  $G$ , the belief base contains the following information:  $pos(G, X, Y)$ , represents the position for  $G$ ;  $files(G, [files])$ , represents the list of files associated to  $G$ ;  $woos(G, N)$ , represent the size, in number of squares, of the window of opportunity for processing  $G$ . The agent’s*

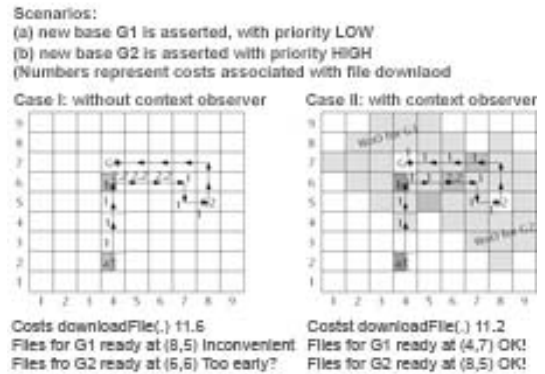


Figure 2: Mobile services in Tileworld

*position is represented as  $pos(X, Y)$ . The agent can simultaneously move one square and download one file per goal per clock period. It can also download files simultaneously, however there is a penalty associated to networking concurrency – set at 10% in this example.*

In one particular run, the scenario is as in the figure. The agent activates the desire  $goBase(G1)$  and the deliberation leads it to commit to that goal. The agent starts to move towards  $G1$ . Let us say that when the agent is at  $pos(4, 6)$ , a new base  $G2$  is asserted with a higher priority. The goals to reach two bases at the same time are conflicting, of course. Hence, the agent has to reconsider the current intention  $goBase(G1)$ , by pausing it and starting the new intention  $goBase(G2)$ . It starts to move towards  $G2$ . The goals to download files are not conflicting, as the agent can download files for different goals concurrently. We aim to show how the context-observer helps to improve the application’s performance in relation to this goal.

In (*Case I*), the application operates without a context-observer. The application starts to download the files for  $G1$  immediately. At  $pos(4, 6)$ , when it learns about  $G2$  (as described above), it continues to download the files for  $G1$  and starts to download the files for  $G2$  concurrently. As there are three files for  $G2$ , it completes this task at  $pos(7, 4)$ , delivering the information at that point. It continues to download the files for  $G1$ , which has nine files associated to it, completing at  $pos(8, 5)$ , that is while the agent is at base  $G2$ .

One can say that this information is delivered at an *inconvenient time*, which is intuitive if, for example, the mobile user is attending to a meeting at  $G2$  and receives the files for  $G1$  at that point. I also note that there are no download operations happening between  $G2$  and  $G1$ , because the goals have been completed already, thus it misses the opportunity to optimise the processing by postponing the downloads for  $G1$  to this convenient time. In addition, the “overhead cost” of concurrent downloads between  $pos(4, 6)$  and  $pos(6, 6)$  results in total costs equals 11.6.

In comparison, in (*Case II*), the application operates with a context-observer. The desire to download the files for  $G1$  is preceded by a condition that it has to be within the area marked as “Window of Opportunity (WoO)  $G1$ ” to deliver the information for  $G1$  at  $pos(4, 4)$ . Then, it starts to download the files.

When the agent reaches position  $pos(6, 6)$ , it is still in the WoO to download the files for  $G1$  and steps inside the WoO to download the files for  $G2$ . The downloads are processed simultaneously, as explained, however they compete for resources – namely, network bandwidth. Next, the agent steps out of the WoO for  $G1$ , at  $pos(7, 6)$ . An event is generated that the condition for the desire to download the  $G1$  files has been falsified. At this point, the deliberation system can stop  $downloadFiles(G1)$ , because it is no longer relevant. It can then *concentrate resources* on other concurrent deliberations – namely, into  $downloadFiles(G2)$ .

Once the agent reaches  $G2$  and the information is delivered, it has the opportunity to *resume* the deliberation of  $goBase(G1)$ . Again, once heading towards base  $G1$ , at  $pos(7, 7)$  it is in the WoO to deliver the information for  $G1$ . Hence, it can resume the deliberation of  $downloadFiles(G1)$ . The remaining files are downloaded and delivered before reaching  $G1$ .

In this case, the information is delivered at convenient times for both  $G1$  and  $G2$ . In addition, as the file download for  $G1$  is postponed to a more convenient time – that is, paused when out of the WoO for  $G1$  and resumed between  $pos(7, 7)$  and  $pos(4, 7)$  – it reduces the amount of concurrent downloads, thus saving “overhead cost”. The resulting total download costs equals 11.2, which is an improvement over (*Case I*)’s deliberation.

Therefore, the agent is saving resources when it is able to pause and resume the file download deliberations in response to events from the context-observer. The optimisation of the number of operations involved is the visible consequence of taking advantage of environmental information to decide “when” to reconsider. The operations are facilitated by the ability of the planning process to handle concurrent deliberations.

### 3. CONCLUSION AND FUTURE WORK

For a mobile device to function as a context-aware application, it is necessary to integrate many of the different services available on it. The application must therefore deal with a large amount of highly dynamic and heterogenous information. We have shown that while agent architectures are a natural choice for use in such environments, their adoption is not without problems. One of those is recognizing “when” is the optimal time to reconsider desires, plans and actions. The context-observer is designed to alleviate this problem by recognizing whether new information impacts current deliberations or executions. If this is the case the agent is “warned”, thus bypassing the deliberation cycle’s inherent delayed action. The agent is more equipped to react to dynamic information using fewer resources when doing so. We use information provided by the agent designer to support these runtime decisions.

The context-observer performs a role of meta-reasoning and we do not argue that it is the only way that one could solve this conundrum. However, we believe that placing this kind of meta-reasoning as a preprocessing module on incoming information is an intuitive solution that makes it easy for the designer, programmer and user to understand.

The next step in researching the context-observer is to demonstrate, with qualitative and quantitative simulations, that the context-observer can assist the application designer to configure a system to suit characteristics of the domain, and can help provide the user with a coherent, pervasive,

mobile application.

### 4. REFERENCES

- [1] R. H. Bordini and J. F. Hübner. Bdi agent programming in agentspeak using *jason* (tutorial paper). In *CLIMA VI*, pages 143–164, 2005.
- [2] M. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.
- [3] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.
- [4] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.
- [5] M. Dastani and J.-J. Meyer. A practical agent programming language. In *Proceedings of the fifth International Workshop on Programming Multi-agent Systems (ProMAS’07)*, 2007.
- [6] A. Greenfield. *Everyware: the dawning age of ubiquitous computing*. New Riders Publishing, 2006.
- [7] K. V. Hindriks, F. S. d. Boer, W. v. d. Hoek, and J.-J. C. Meyer. Agent programming in 3ap1. *Autonomous Agents and Multi-Agent Systems*, 2:357–401, 1999.
- [8] N. Howden, R. Rönquist, A. Hodgson, and A. Lucas. Jack intelligent agents - summary of an agent infrastructure. In *5th International Conference on Autonomous Agents*, 2001.
- [9] F. Koch and I. Rahwan. The role of agents in intelligent mobile services. In *Proceedings of the 7th Pacific Rim International Workshop on Multi-Agents (PRIMA)*, volume 3371, Berlin, Germany, 2005. Springer-Verlag.
- [10] M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*. AgentLink, 2005.
- [11] M. O’Grady and G. O’Hare. Mobile devices and intelligent agents  $\dot{U}$  towards a new generation of applications and services. *Intelligent Embedded Agents*, 171:335–353, May 2005.
- [12] A. S. Rao and M. Georgeff. Bdi agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems, ICMAS 95*, 1995.
- [13] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR’91)*, pages 473–484, San Mateo, CA, USA, 1991. Morgan Kaufmann publishers Inc.
- [14] M. Weiser. The computer for the twenty-first century. *Scientific American*, pages 94–104, 1991.
- [15] M. Wooldridge and N. R. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.