# Programming Deliberative Agents for Mobile Services: the 3APL-M Platform.

Fernando Koch[1], John-Jules C. Meyer[1], Frank Dignum[1], and Iyad Rahwan[2]

[1] Institute of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
fkoch@acm.org, {jj,dignum}@cs.uu.nl
[2] Institute of Informatics, The British University in Dubai
P.O.Box 502216, Dubai, UAE
iyad.rahwan@buid.ac.ae

**Abstract.** 3APL-M is a platform for building deliberative multi-agent systems whose components execute on handheld and embedded computational devices. The solution takes advantage of the 3APL language and definitions, delivers a methodology for building Belief-Desire-Intention inference systems and provides an interface to integrate the applications to the external world. The library is distributed for the Java 2 Micro Edition (J2ME) programming platform, which is widely adopted by the hardware manufactures and available for a myriad of mobile computing devices. The role of agent-based computing for mobile services is explained, the architecture and programming structures are presented and proof-of-concept applications are demonstrated.

## 1 Introduction

The promise of mobile technologies is to remove the bindings between a fixed space and a person's information and communication resources [19]. Intelligent mobile services should make use of local processing to reason about the user's context and predict user's intents, actions and location. However, mobile computing introduces issues of resource limitations, security, connectivity and, limited power supply, which are inherent to the environment [21]. These characteristics call for the optimal use of local resources, communications and connectivity. Therefore, the problem is how to create intelligent mobile applications that execute on mobile computing devices.

Agent-based computing [14][23] seems to offer a set of features that are very closely aligned with the requirements of service delivery challenge in mobile computing [16]. For the purpose of this paper, agents are computer systems capable of flexible autonomous action in dynamic, unpredictable and open environment [17].

This paper presents the 3APL-M (Triple-A-P-L-M) platform for implementing deliberative autonomous agents that execute on mobile computing devices. It works as a scaled down implementation of the 3APL language interpreter [12][8] re-designed for the requirements of mobile computing applications. The inference

system implements the Belief-Desire-Intention paradigm [20], which intrinsically provides the solutions for designing systems capable to creating mental models [7][18]. Moreover, it supplies the programming structures to implement *sensors* for context-sensitiveness [10][22] and *actuators* for pervasive content delivery.

The paper is organized as follows. Section 2 **analyses** the use agent-based computing for the development of mobile service applications. Next, section 3 presents our **approach** for building the scaled down version of the 3APL platform. Section 4 presents the **solution** for the 3APL-M system architecture. Finally, section 5 presents the **results**, as two proof-of-concept applications implemented using the platform and their running parameters. The paper concludes by presenting the achieved results and pointing to further works.

## 2 Agents and Mobile Service

The role of agent-based computing in mobile services is to provide solutions for the requirements of the future software systems, as presented in [25] [24]:

- *situatedness*, as the mobile service must be aware of the environmental conditions surrounding the mobile user;
- *openness*, as the mobile service's components must be able to integrate and adapt to the presence of new modules being integrated to or removed from the system's environment;
- *local interaction*, as the mobile service's applications and components must be able to interact to other modules and interact with the environment, and;
- *local control*, related to the problem of implementing mobile applications able to run autonomously.

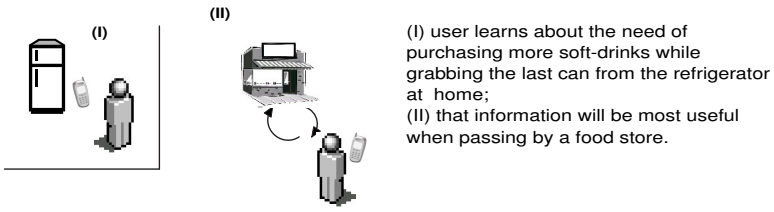



**Fig. 1.** Mobile Commerce Scenario

For the sake of demonstration, let us consider the scenario shown in Figure 1:

- (I) the user enters his shopping list at home, in front of his fridge when running out of a product.

– (II) when the user is walking by a grocery store, the location-based service detects the user's position and notifies the local processing application. This application holds the user data and has the capability of negotiating the stored shopping list. Several aspects of the context could be taken in consideration during the deliberation. For example, the user's agenda (the negotiation should be avoid if the user has an appointment setup for the next minutes); the user's preferred stores (the application should be able to collect the quote from the stores where the user normally does its shopping); availability of computing resource (avoid the negotiation if the device is running low in power supply), and connectivity.

The requirements to implement this mobile solution are: the structures for knowledge representation (shopping list, preferred stores, calendar and device information); the interface to a location detection system; an inference system that cross relate the internal and context information; a negotiation system, and; a content delivery interface.

Agent-based software engineering provides the tools to implement these requirements, as presented in [16]. The solutions provided by the agent-paradigm are:

– *Structures for knowledge representation*: existing agent systems can provide an answer to the situadedness requirement. This ability is an intrinsic problem in multi-agent systems, and hence inherent in agent architectures, especially in the belief-desire-intention paradigm. In the demonstration, it provides the structures to represent the shopping list, preferred stores, device information and calendar.
– *Responsiveness and adaptivity*: as pointed out in [14], these are inherent features provided by agent systems; agents should be able to adapt to constantly changing execution environment. In the demonstration, it provides the features to either dropping or adapting the negotiation process in answer to the computing resource availability information.
– *Sociability and locality of interaction*: also described in [14], agents are able to interact with other agents or humans when needed. In the demonstration, this feature would provide the support for the negotiation process.
– *Autonomy*: as argued in [15], the agent paradigm offers mechanisms that address varying degrees of autonomy, from basic reactive architectures based on a set of pre-determined rules, to mechanisms for proactive behaviour [9] considering the context and user preferences. In the demonstration, the local processing agent must be able to act autonomously for adapting the application execution in face to possible computing or connectivity problems.

Moreover, agent-based software engineering incorporates support for *decomposition*, *modularity* and *abstraction* [13], which are essential features considering the distributed nature of mobile computing applications.

## 3 Approach

A platform for building agents in mobile devices must provide solutions for the problems inherent to the environment, such as computing resource availability, networking, security, interfacing and compatibility. For example, how to execute the deliberation cycle in the limited computing resources environment?; how to implement the structures for context awareness and content delivery?; what agent-oriented language to use for the development of the application knowledge structures?; which programming language to choose for the application development?

The requirements for the development of 3APL-M are to be:

- compatible with 3APL language and programming environment;
- lightweight enough to be deployed on small devices with as few as 20Mhz CPU and 512Kb RAM.
- developed in the Java 2 Micro Edition (J2ME) [4][11] programming platform. J2ME is a reduced version of the Java programming platform tailored to fit in low profile mobile computing devices. It provides a programming and runtime environment for Java coded applications. This environment is widely adopted by the hardware manufactures and available for a myriad of mobile computing devices;
- optimized for processing, reducing the number of operations per deliberation, thus ensuring performance and minimum battery utilization, and;
- provide the application programming interface (API) for the integration of the 3APL application to context-awareness and content-provider structures.

### 3.1 About 3APL

3APL is a logic-based agent programming language that provides constructs for implementing agents' beliefs, plans and capabilities as explicit run-time entities. It uses practical reasoning rules in order to generate plans (i.e., sequence of actions) for achieving the applications goals. Each 3APL program is executed by means of an interpreter that deliberates on the cognitive attitudes of that agent. More information about the 3APL language, syntax and logic fundaments is available at the 3APL project's web-site at [2].

Figure 2 presents the abstract architecture of 3APL. Each agent has the explicit representations of its goals in the *goal base*. For example, the goal to finish an assignment may be represented with the predicate *finish*(*assignment*). In order to achieve its goals, the agent decomposes these into *sub-goals* using planning rules from the *plan rule base*. The sub-goals can be further decomposed until *basic actions* are reached (i.e., physical actions agents may execute directly in the world).

During plan generation, the agent takes into account its *belief base*, which stores the contextual information in form of predicates. For example, the predicate *near*(*fernando*, *storeA*) denotes that the agent believes *Fernando* is currently located near the *storeA*. The *capability base* describes basic actions by the agent

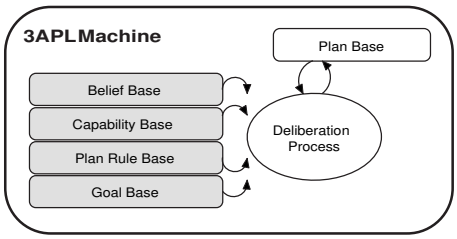



**Fig. 2.** 3APL Architecture

and user. A planning rule takes the form $head \leftarrow guard|body$, and means that if the agent has goal $g$ that unifies to the head of the plan *head* and the condition declared in *guard* is satisfied (i.e., it unifies to the contents of the belief base), then goal $g$ can be achieved by executing the sequence of actions (or set of sub-goals) listed in *body*.

As it will be presented in the next section, the application architecture is influenced by the features of the 3APL language and the platform requirements.

## 4 System Architecture

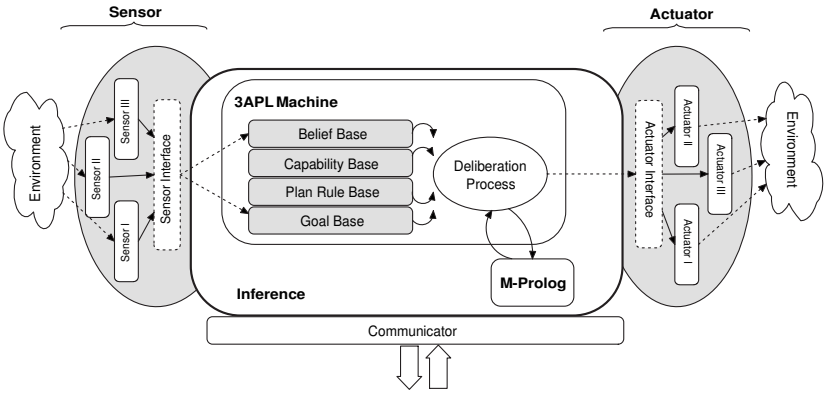



**Fig. 3.** 3APL-M Architecture

The 3APL-M platform architecture is presented in Figure 3. The main features are: sensor and actuator modules, which provide the interface to integrate to context-awareness and content delivery solutions; the 3APL machinery, which includes the infrastructures for the B.D.I. based inference systems, and; the communicator module, which provides the support for communication in a multi-agent system. The modules are explained below:

– the *3APL machine* encapsulates the 3APL language components and provide the programming interface for the integration of the logic structures to the Java programming language;
– the *belief*, *capabilities*, *goal* and *plan rules* modules are implementations of the 3APL structures. These elements are part of the 3APL machinery and provide the internal data and processing structures for the platform;
– the *deliberation process* is the implementation of the executive module (deliberation cycle);
– the *plan base* is the data structure that holds the list of current plans generated by the deliberation process;
– the *plan ranker* is the sub-routine that, if enabled, classifies the plan bases during the deliberation cycle and order the plans based on their *utilities* (difference between the worth of the goal and the costs of the actions).
– the *m-prolog* is an optimized implementation of the PROLOG language engine, used for the low-level inference processing in 3APL-M. It also holds the data for the belief base. The m-prolog is a sub-product of this project and it is packaged along the distribution library.
– the *sensor* and *actuator* are the programming interfaces for the integration of the 3APL-M machinery to the external world. The *sensor* module provides the infrastructure for the creation of context-aware application (i.e. environmental sensors) and system input (i.e., device's keyboard). The *actuator* module provides the means for content delivery (i.e., integration to the device's display interface) and acting upon the environment.
– the communicator module provides the is the generic interface for the data exchange infrastructure, required for multi-agent system module integration and communication to external services. The module provides internal support for FIPA communication [5][6], however any other protocol or data representation can be plugged in the system through a plug-in interface.

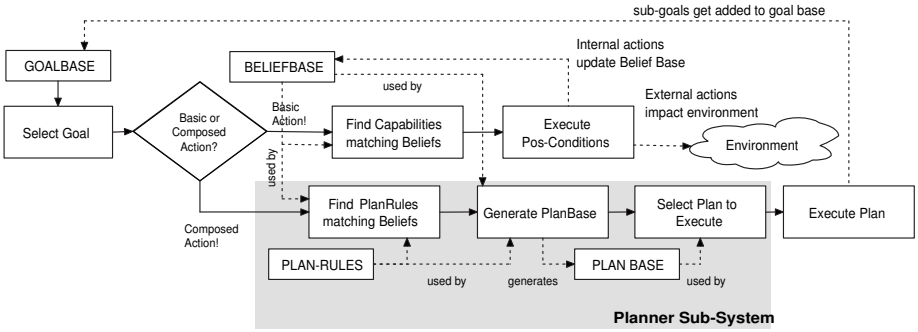



**Fig. 4.** Basic Deliberation Cycle

For the *deliberation process*, the 3APL-M platform provides the Java programming class *Agent*, which implements the *basic deliberation cycle* [12]. Due

to space limitation, this work shall not discuss the deliberation process in detail but introduce the general idea. For detailed information, we refer to Hindriks et al [12] and Dastani et al [9].

The *basic deliberation cycle* is depicted in Figure 4. In this case, the agents generate their plans by choosing the first applicable rule that matches a particular goal/desire. This means that an agent generates only one plan for each achievable goal, and only generates other plans if the initial plan fails.

### 4.1 Programming

The 3APL-M platform works as a library loaded in the distribution package. This library supplies the application-programming interface (API) for the 3APL machine modules. The Java application make calls to the library's modules for loading information, configuring the deliberation engine and executing the application. Figure 5 presents: (A) the 3APL-M programming interface for the Agent class, and; (B) a simple Hello World Java-3APL-M code example.

**(A) Agent class programming interface (simplified view)**

```
void addBelief(String beliefStr) // Add a belief.
void addCapability(String capabilityStr) // Add a capability
void addGoal(String goalStr) // Add a goal
void addPlanRule(String planRuleStr) // Add a plan
void addProlog(String prologStr) // Add Prolog knowledge
void addActuator(String actionStr,
        ActuatorInterface actuator) // Add actuator
void addSensor(String id, Sensor sensor, int interval,
        boolean addGoalNotification) // Add Sensor
void deliberate() // Starts deliberation cycle
void destroy() // Terminate agent
String sendMessage(String msgId, String to,
        String performative, String data) // Send a message
void setFipaCommunication(boolean enabled)
```

**(B) HelloWorldsource code**

```
public class HelloWorldExample {
 public void startApp () {
  Agent ag = new Agent("hello");

  // load knowledge
  ag.addCapability ("{} Print(X) { GUI(print, X)}");
  ag.addPlanRule (" <- TRUE | Print('hello world')");
  ag.addGoal ("print");

  // add J2ME display actuator
  ag.addActuator ("GUI(Type,Message)",
      new J2MEGUI(this));

  // deliberate
  ag.deliberate ();
 }
}
```

**Fig. 5.** Programming with 3APL-M

Figure 5(B) presents an example for the programming steps. The code must instantiate a new *Agent* object and to load the 3APL information (i.e., beliefs capabilities, goals, plan rules) using the Agent methods (presented in Figure 5(A)). Next, sensors and actuators can be initialized and attached using the *addSensor(.)* and *addActuator(.)* methods. Finally, the deliberation process is started by calling the *deliberate()* method.

For detailed information about programming in 3APL-M, we refer to the documentation and source code examples available at the project's web-site [1].

## 5 Results

This section presents two proof-of-concept implementations using 3APL-M platform. These are simple applications aiming to present the programming struc-

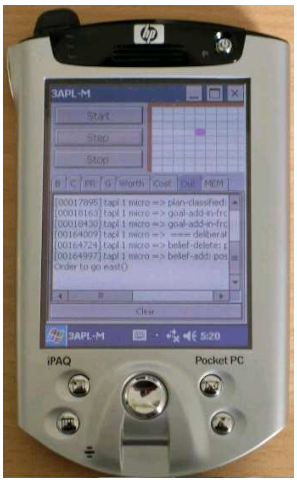
tures, running parameters and integration of Java and 3APL code. The source code for these and other demonstration applications can be found at the project's web-site [1].

### 5.1 Block World Demonstration Application

The Block world demonstration is presented to show the compatibility between the 3APL-M and the 3APL standard specifications. This is the example provided at the 3APL web-site [2].

The application is composed by a robot that needs to arrive to a base in a grid world. The robot knows where are the bases and the rules for the decision process. The knowledge representation and deliberation process is implemented in 3APL and the GUI manipulation is done in Java. Figure 6 presents: (A) the application running on a HP iPaq hardware; (B) the 3APL code, and; (C) the 3APL-M and Java code integration.

**(A) BlockWorldon HP iPaq**




**(B) 3APL code for robot deliberation**

```
CAPABILITIES:
{pos(X, Y)} West { NOT   pos(X, Y), pos(X - 1,  Y), BlockMove(west)}.
{pos(X, Y)} East { NOT   pos(X, Y), pos(X + 1,  Y), BlockMove(east)}.
{pos(X, Y)} North { NOT   pos(X, Y), pos(X, Y + 1),  BlockMove(north)}.
{pos(X, Y)} South { NOT   pos(X, Y), pos(X, Y - 1),  BlockMove(south)}.
{}  BlockMove(X) {EXTERNAL}.

RULEBASE
goBase <-  pos(X, Y) AND base(X,  Y) | SKIP.
goBase <-  pos(X, Y) AND base(A, B) AND X > A | West,    goBase.
goBase <-  pos(X, Y) AND base(A, B) AND X < A | East,    goBase.
goBase <-  pos(X, Y) AND base(A, B) AND  Y > B | South,   goBase.
goBase <-  pos(X, Y) AND base(A, B) AND  Y < B | North,   goBase.

BELIEFBASE
pos(9, 9).
base(0, 0).

GOALBASE
goBase.
```

**(C) 3APL-M and Java integration**

```
/**
 * Midlet Interface
 */
public void  startApp () {
  // create agent
  Agent  ag = new Agent("robot");
  // load knowledge bases
  ag.loadfromStream  (System.getResourceAsStream   ("robotAgent.tapl   "));
  // attach actuator
  ag.addActuator   ("BlockMove(X)", new  BlockWorldActuator  (this.blockWorld  ));
  // deliberate
  ag.deliberate  ();
}
```

**Fig. 6.** BlockWorld demonstration interface and code

In this example, the Java code initializes the agent (new *Agent(.)*), loads the knowledge (3APL code) from an input stream (*ag.consult(.)*) and attaches the Block World interface actuator (*ag.addActuator(.)*). Next, the application triggers the deliberation process (*ag.deliberate()*). The 3APL machinery will load the intention from the goal base (*goBase*). From the deliberation, the 3APL code

will end up calling the Block world actuator passing the argument "west". The Java coded *BlockWorldActuator.process("west")* will be executed to update the interface.

On the HP iPaq device, this application executes using 142.7 Kbytes of RAM memory and takes approximately eight seconds to find a solution (including interface update time). In total, it process 38 deliberation steps and requires 539 unifications operations on the PROLOG engine.

### 5.2 Mobile Commerce Demonstration Application

This demonstration presents the 3APL-M based implementation for the mobile commerce problem from Figure 1. For simplicity, the demonstration will concentrate on the 3APL code and Java integration and overlook technical details about the location-based service and connectivity. It is assumed that there is a location-based service feeding the agent's belief base with landmark proximity information and there is stable connectivity.

**(A) Screen shots**


**(B) 3APL code**

```
CAPABILITIES:
{ shoppingList (List)}  AddItemToList (Item)
{ NOT  shoppingList (List),  shoppingList (List + Item)}.
{}  AskConfirmation (Message) { GUI(promptYesNo , Message)}.
{} Display(Message) {   GUI(promptOk , Message)}.
{}  GUI(Type, Message) {EXTERNAL}.

RULEBASE
addItemToList (Item) <- TRUE |
AddItemToList (Item).

displayQuote (Shopping, Quote) <- TRUE |
Display([Quote received from , Shopping,  is $, Quote]).

getQuote (Shopping, List, Result) <- TRUE |
Send( MsgId , Shopping, query-  ref , quote(List)),
Receive( MsgId , Shopping,  Performative  , Result, 4).

resolve <- location(near, Shopping) AND     shoppingList (List) |
AskConfirmation ([Near , Shopping, . Request for quote?]),
getQuote (Shopping, List, Result),
Assert( receivedQuote (Shopping, List, Result)),
displayQuote (Shopping, Result).

BELIEFBASE
addressBook (storeA , http://  localhost :50001).
shoppingList ([ productA ,  productB ]).
location(near,   storeA ).
```

**Fig. 7.** Mobile Commerce Solution: (A) Conceptual Model and (B) 3APL code

The 3APL code for this solution is presented in Figure 7(B) and the screenshots from the running application in a mobile phone simulator are depicted in Figure 7(A).

Basically, when a landmark proximity is detected (near grocery store), the location service provider adds the context information to the agent's belief base

(*location(near, storeA)*), the goal resolve to the goal base and starts the deliberation process. The sequence of actions will be created by processing the plan rule named *resolve* if there is a *location(.)* and *shoppingList(.)* available in the belief base. The sequence of actions are: to ask the confirmation on the negotiation process to the user (*AskConfirmation(.)*); in case of positive answer, to request the quote from the store (*getQuote(.)*); once the quote is received, to assert that information in the belief base (*Assert(receivedQuote(.)*), and; finally, to display the received quote in the devices interface (*displayQuote(.)*).

From the sequence of actions above, some will be decomposed in sub-goals and added to the goal base while others will trigger capabilities. The capabilities are executed based on the definition in the capability base, built-in capabilities (e.g., *Assert(.)*, *Send(.)*, *Receive(.)*) or through attached actuators (e.g., *GUI(.)*). For a complete list of the built-in capabilities, refer to the documentation available in the project's web-site [1].

Once again, this is a simplified demonstration application aiming as a proof-of-concept and several improvements are possible. The demonstration run on the phone simulator supplied in the J2ME Wireless Toolkit 2.1, from Sun Corporation. The solution utilizes 163.8 Kbytes of RAM memory and processes eight deliberation.

## 6 Conclusion

3APL-M provides the support technology to develop deliberative multi-agent systems to be executed in mobile computing devices. The main features are the *sensor* and *actuator* modules, which provide the interface to integrate to context-awareness and content delivery solutions; the 3APL machinery, which includes the infrastructures for the B.D.I. based inference systems, and; the communicator module, which provides the support for communication in a multi-agent system. Hence, the platform provides the infrastructures the technologies for the new generation of mobile application: context-sensitiveness, mental modelling, local processing, and pervasive content delivery. The B.D.I.-based inference inherently provides the solutions for designing systems capable to creating mental models, represent the human thought structures.

The platform delivers a development environment compatible with the 3APL language structures. The demonstration applications show that the resulting applications are reduced enough to be deployed on small devices with 20Mhz CPU and less than 512Kb RAM. The platform is compatible with Java 2 Micro Edition (J2ME) development and running environment, which has a large development community, and, consequently, several development environments, platforms and programming libraries are commercially available. The strength of J2ME is industry adoption and to be Java-compatible. This running environment is present in a myriad of mobile computing devices today and this number tends to grow in the near future.

There are several possible enhancements and optimizations for the platform. A future line of work is to better position the project against *FIPA standards* [3]

especially for communication and community management. Moreover, *security* is a major area of research to be explored by this project. While there are limitations already imposed by the running environment – e.g., Java 2 Micro Edition sandbox security – the high-level security must be implemented by means of platform structures and logic operations.

For detailed information about programming in 3APL-M, downloads, demonstration codes and documentation, refer to the project's web-site [1].

## Acknowledgments

This work was conducted while Fernando Koch and Iyad Rahwan were working at the Department of Information Systems, University of Melbourne. The authors are grateful to the Department of Information Systems and Hewlett Packard's Philanthropy Division for providing the test equipment and for the discussions surrounding the paper topic.

## References

1. 3APL-M web-site, http://www.cs.uu.nl/3apl-m.
2. 3APL web-site, http://www.cs.uu.nl/3apl.
3. Foundation for intelligent physical agents (FIPA) web-site, http://www.fipa.org.
4. Java 2 Micro Edition web-site, sun corporation, http://java.sun.com/j2me.
5. M. Aparicio, L. Chiariglione, E. Mamdani, F. McCabe, R. Nicol, D. Steiner, and H. Suguri. FIPA - intelligent agents from theory to practice. *Telecom 99*, October 1999.
6. F. Bergenti, A. Poggi, B. Burg, and G. Claire. Deploying FIPA-compliant systems on handheld devices. *IEEE Internet Computing*, 5(4):20–25, 2001.
7. Z. Chen. Building expert systems through the integration of mental models. In *IEA/AIE '88: Proceedings of the first international conference on Industrial and engineering applications of artificial intelligence and expert systems*, pages 754–761. ACM Press, 1988.
8. M. Dastani, F. de Boer, F. Dignum, and J.-J. Meyer. Programming agent deliberation: An approach illustrated using the 3APL language. In *Autonomous Agents and Multi-Agent Systems 2003 (AAMAS'03)*, Melbourne, Australia, 2003.
9. M. Dastani, F. Dignum, and J.-J. Meyer. Autonomy and agent deliberation. In M. Rovatsos and M. Nickles, editors, *The First International Workshop on Computatinal Autonomy - Potential, Risks, Solutions (Autonomous 2003)*, pages 23–35, Melbourne, Australia, July 2003.
10. A. K. Dey. *Providing Architectural Support for Building Context-Aware Applications.* PhD thesis, Georgia Institute of Technology, November 2000.
11. E. Guigere. *Java 2 Micro edition: The ultimate guide on programming handheld and embedded devices.* John Wiley and Sons, Inc., USA, 2001.
12. K. V. Hindriks, F. S. De Boer, W. Van Der Hoek, and J.-J. Ch. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.
13. N. R. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.

14. N. R. Jennings and M. Wooldridge. Applications of intelligent agents. pages 3–28, 1998.
15. F. Koch and I. Rahwan. Classification of agents-based mobile assistants. In *Proceedings of the AAMAS Workshop on Agents for Ubiquitous Computing (UbiAgents)*, New York, USA, Jul 2004.
16. F. Koch and I. Rahwan. The role of agents in mobile services. In *Proceedings of the Pacific Rim International Workshop on Multi-Agents (PRIMA2004)*, Auckland, NZ, August 2004.
17. M. Luck, P. McBurney, and C. Preist. *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*. AgentLink, 2003.
18. B. Mulder. The role of mental models in designing computer systems. Master's thesis, Vrije Universiteit, Amsterdam, 2000.
19. M. Perry, K. O'Hara, A. Sellen, B. Brown, and R. Harper. Dealing with mobility: understanding access anytime, anywhere. *ACM Transactions on Computer-Human Interaction*, 8(4):323–347, 2001.
20. 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.
21. M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17, 2001.
22. B. N. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proc. of IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, California, USA, Dec 1994.
23. M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, jun 1995.
24. F. Zambonelli and H. V. D. Parunak. Towards a paradigm change in computer science and software engineering: a synthesis. *The Knowledge Engineering Review*, 2004. (to appear).
25. F. Zambonelli and H. Van Dyke Parunak. From design to intention: Signs of a revolution. In *Proceedings of the First International Joint Conference on Autonomous agents and Multiagent Systems*, pages 455–456. ACM Press, 2002.