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. 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 [26]. 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 [16] seems to offer a set of features that are very closely aligned with the requirements of service delivery challenge in mobile computing [18]. For the purpose of this paper, agents are computer systems capable of flexible autonomous action in dynamic, unpredictable and open environments.

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 [14] re-designed for the requirements of mobile computing applications. The inference system implements the Belief-Desire-Intention paradigm [25], which intrinsically provides the solutions for designing systems capable to creating mental models. Moreover, it supplies the programming structures to implement *sensors* for context-sensitiveness [12] 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 Motivation and Related Work

In this section, we introduce the role of agent-based computing in delivering mobile services and present the related works that deliver a platform to build those applications.

The role of agents in mobile services is to provide the support to the requirements of the future generation of software applications [27]. The support provided by agents are in the realms of:

- *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.

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

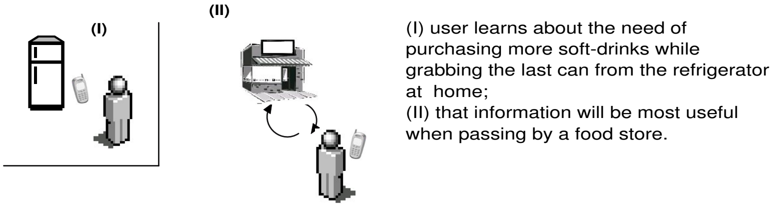

(I)
(II)
(I) user learns about the need of purchasing more soft-drinks while grabbing the last can from the refrigerator at home;
(II) that information will be most useful when passing by a food store.

Fig. 1. Mobile Commerce Scenario

the stored shopping list. Several aspects of the context could be taken into consideration during the deliberation. For example, the user's agenda (the negotiation should be avoided if the user has an appointment set up 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 [18]. 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 [16], 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 [16], 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 [17], 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 [11] 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* [15], which are essential features considering the distributed nature of mobile computing applications.

2.1 Related Work

Here we introduce the related works that deliver platforms to build agents-based applications in mobile computing devices. In [21] it is argued that making agents to run in resource-constrained devices is still not an obvious task. We have selected a number of available platforms, which we considered to be representative of what is available.

In [8] it is presented a model of agent construction for ubiquitous computing which is conceptually grounded and architecture neutral and makes use of a component based approach for agent design. The project uses *S.M.A.R.T.* (Structural, Modular agent Relationship and Types) framework and *actSMART*[9] for the implementation in a Java 2 Micro Edition [13] platform. Although the work presents a support to generic services in ubiquitous computing environment, it does not focus on the problem of supporting the development of intelligent personal assistants.

The Lightweight Extensible Agent Platform (LEAP) [10],is the first attempt to implement a FIPA [3] agent platform that runs seamlessly on both mobile and fixed devices over both wireless and wired networks. It uses a set of profiles that allows one to configure it for execution on various machines, OS and Java VM. This platform has many strengths and satisfies the requirements for intelligent support (B.D.I. based), collaboration and personal assistance. Although the platform can be adapted to integrate to, e.g., context-awareness and device interface support, this feature is not clearly defined in the product.

The MobiAgent platform [22] delivers a solution where neither the platform nor the agents run locally in the device. In this solution, when the user wants to delegate a task to an agent, the mobile device connects to the Agent Gateway and downloads an interface that configures it. The agent performs its task and, later, reports the results via the same mechanism. The shortcoming of this solution is the dependency of a reliable connectivity mechanism between the device and the Agent Gateway.

The kSACI platform [5] is a smaller version of the SACI platform [6]. SACI is an infrastructure for creating agents that are able to communicate using KQML [19][20] messages and use a mailbox structure to exchange messages. Although kSACI platform is usable on small devices running the Java 2 Micro Edition, the platform is not entirely situated on the small device. Moreover the kSACI is oriented to communication aspects of multi-agent system and does not provision for enhanced inference systems.

Finally, the *AbIMA* platform [23] delivers agent-based intelligent mobile assistant that runs on a handheld device and assists the user through the execution of individual tasks. It makes use of the abstract agent programming language AgentSpeak(L) [24]. Nevertheless, AbIMA offers support to single-user environments only.

Table 1 summarises support provided by the aforementioned platforms to the components in intelligent mobile service solutions: (i) *Local Processing* is the support to local execution for personal assistant applications; (ii) *Context Awareness* is the support to the component for context awareness in mobile applications; (iii) *Inference* is the support to the component for "enhanced deliberation" in intelligent applications; (iv) *Collaboration* is the support to the component for collaboration in multi-user environment, and; (v) *Device Interface* is the support to interfacing in mobile computing.

Table 1. Classification of Platforms for Agent-Based Applications in Mobile Computing

Classification of Agent-based Platforms for Mobile Computing and Mobile Personal Assistant Solutions

Platform	**(i) Local Processing**	**(ii) Ctx. Awareness**	**(iii) Infer.**	**(iv) Collab.**	**(v) Dev. Interface**
JADE/LEAP	Yes (various)	Not Explicit	Yes (BDI)	Yes (FIPA)	Not Explicit
MobiAgent	No	No	No	Yes	Partial
kSACI	Yes (J2ME)	No	No	Yes	Not Explicit
AbIMA/ AgentSpeak(L)	Yes	Not Explicit	Yes (BDI)	No	Not Configurable

Hence, based on the analysis of the related work, we conclude that there is a unaddressed opportunity to deliver a platform that supports the components required for in intelligent mobile service solutions. In the next sections, we move towards the specifications for a platform to build B.D.I. architecture, agent-based applications in mobile computing devices.

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][13] 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.

The resulting 3APL-M implementation is fully compatible with the 3APL language and syntax. It is a "cut down" version of 3APL, with the structures optimized for the creation of mobile service applications and deployment in mobile computing devices. Nevertheless, the platform delivers a solution as powerful as the original 3APL implementations. In fact, when executed in the desktop environment, this platform can be an alternative for 3APL solution implementations.

In the next sub-sections, we will introduce the 3APL programming language and then present the system architecture for 3APL-M platform.

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 fundament is available at the 3APL project's web-site at [2].

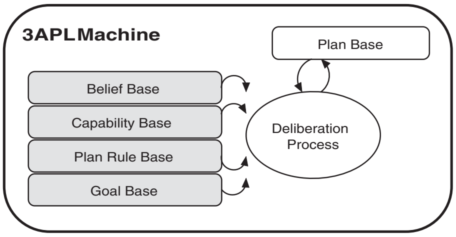


Fig. 2. 3APL Architecture

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 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

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 in the 3APL-M architecture are explained below:

- the *3APL machine* encapsulates the 3APL language components and provides the programming interface for the integration of the logic structures to the Java programming language. This module provides a runtime interpreter for the complete semantics of the 3APL 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 *m-prolog* is an implementation of the PROLOG language engine, optimized to be used for the low-level inference processing in 3APL-M. The m-prolog programming interface holds special structures to make it more compatible to 3APL engine programming. However, it is a fully compatible PROLOG language implementation and, in fact, PROLOG applications can be executed in this environment.

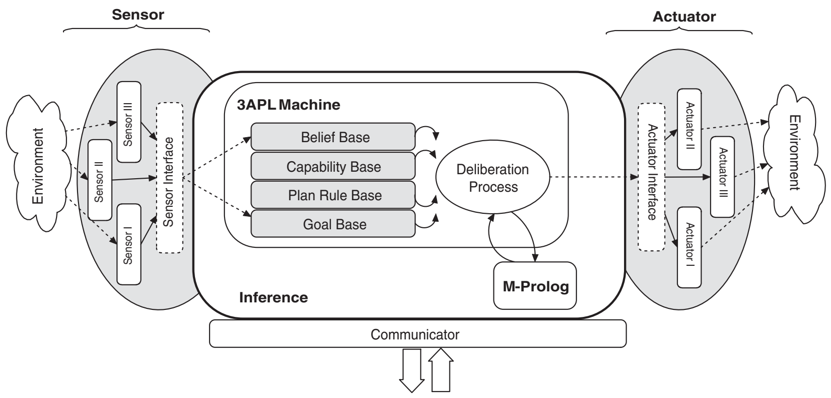


Fig. 3. 3APL-M Architecture

- 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 [7][10], however any other protocol or data representation can be plugged in the system through the programming interface.

The 3APL-M architecture emphasizes the *sensor module* as the input interface for data from the external world. Popular BDI models have neglected "perceptions" as the mental state component that is the basis of communication and interaction. In the classic BDI architecture data is collected by some interface structure and inserted into the BDI belief base. In some BDI approaches, perceptions are indeed treated as beliefs. However, this is clearly unsatisfactory, both conceptually and technically. Conceptually, perceptions are transient, while beliefs are persistent. Hence, the introduction of a *sensor* module provides the technical support to map the perception of an event can into a corresponding event has happened belief, thus avoiding a irrelevant perceptions that would lead to an overflow of the belief/knowledge base of an agent.

Moreover, for the *deliberation process*, the 3APL-M platform provides the Java programming class *Agent*, which implements the *basic deliberation cycle* [14]. 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 [14] and Dastani et al [11].

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 par-

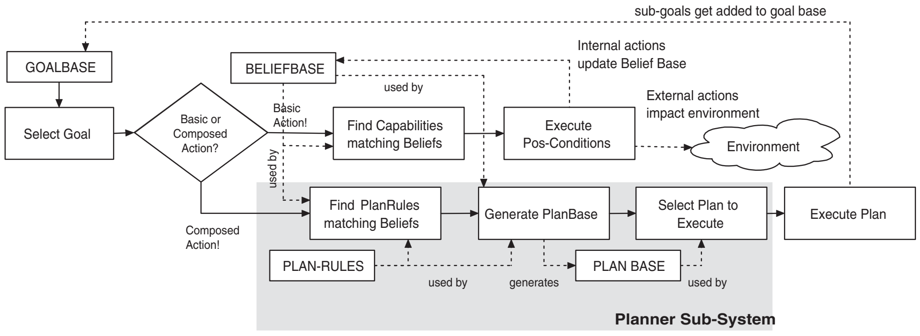
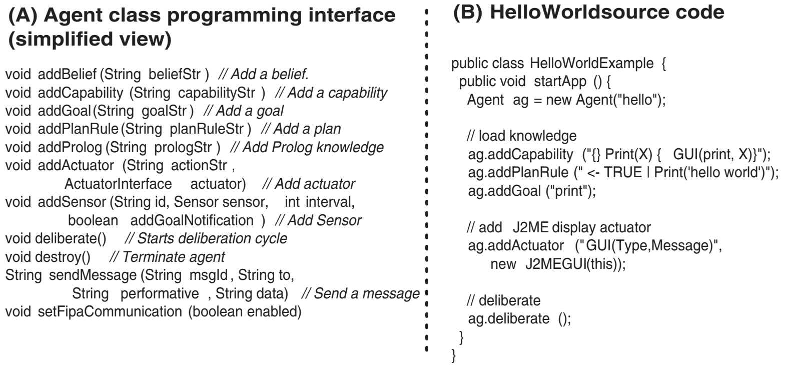


Fig. 4. Basic Deliberation Cycle

ticular 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 makes calls to the library's modules for loading information, configuring the deliberation engine and executing the applications. Figure 5 presents: (A) the 3APL-M programming interface for the *Agent* class (simplified view), 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 structures, 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

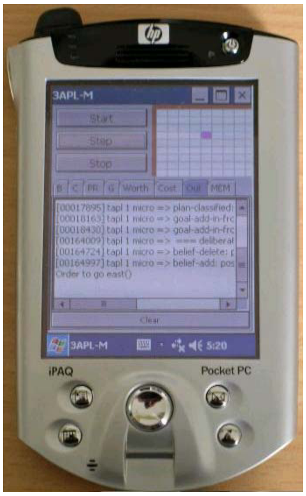
The Block world demonstration is presented to show the compatibility between the 3APL-M and the 3APL standard specifications. This is the example provided at the 3APL web-site [2].

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.

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.actuator(["west"])* will be executed to update the interface.

The test was executed using 3APL-M version 1.3. 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

(A) BlockWorldon HP iPaq



```
[00017895] tapl 1 micro => plan-classified:
[00018163] tapl 1 micro => goal-add-in-frc
[00018430] tapl 1 micro => goal-add-in-frc
[00164009] tapl 1 micro => === deliberat
[00164724] tapl 1 micro => belief-delete: p
[00164997] tapl 1 micro => belief-add: pos
Order to go east()
```

(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.consult (System.getResourceAsStream ("robotAgent.tapl "));
  // attach actuator
  ag.addActuator ("BlockMove(X)", new BlockWorldActuator (this.blockWorld ));
  // deliberate
  ag.deliberate ();
}
```

Fig. 6. BlockWorld demonstration interface and code

total, it process 38 deliberation steps and requires 539 unifications operations on the PROLOG engine.

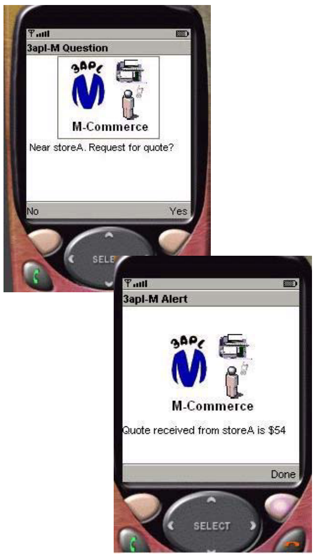
5.2 Mobile Commerce Demonstration Application

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.

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(.)*).

(A) Screen shots


3apl-M Question

Near storeA. Request for quote?

No Yes

3apl-M Alert

Quote received from storeA is $54

Done

(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

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, we refer to the documentation available in the project's web-site [1].

Once again, this is a simplified demonstration application and several improvements are possible. The test was executed using 3APL-M version 1.3 and run on the phone simulator supplied in the J2ME Wireless Toolkit 2.1, from Sun Corporation. The execution utilized 163.8 Kbytes of RAM memory and processed eight deliberation steps.

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 for the technologies required by the new generation of mobile applications: context-sensitiveness, mental modelling, local processing, and pervasive content delivery. The B.D.I.-based inference module provides the solutions for applications capable of creating mental models and to represent the human thought structures.

The platform delivers a development environment compatible with the 3APL language structures. The demonstration applications proved that the resulting applications are small 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. Consequently, several development environments, platforms and programming libraries are commercially available. The strength of J2ME is industry adoption and to be Java-compatible, thus this running environment is present in a myriad of commercially available mobile computing devices.

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, we 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 thankful to the folks in that department 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 (J2ME) web-site, sun corporation, http://java.sun.com/j2me.
5. kSACI web-site, http://www.cesar.org.br/ rla2/ksaci/.
6. Simple Agent Communication Infrastructure (SACI) web-site, http://www.lti.pcs.usp.br/.
7. 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.
8. R. Ashri and M. Luck. An agent construction model for ubiquitous computing devices. In *Proceedings of AAMAS Workshop in Agent Oriented Software Engineering*, New York, USA, 2004.
9. R. Ashri, M. Luck, and M. d'Inverno. actsmart - building a smart system. In M. d'Inverno and M. Luck, editors, *Understanding Agent Systems.* Springer-Verlag, 2nd edition edition, 2003.
10. F. Bergenti, A. Poggi, B. Burg, and G. Claire. Deploying FIPA-compliant systems on handheld devices. *IEEE Internet Computing*, 5(4):20–25, 2001.
11. 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.
12. A. K. Dey. *Providing Architectural Support for Building Context-Aware Applications.* PhD thesis, Georgia Institute of Technology, November 2000.
13. E. Guigere. *Java 2 Micro edition: The ultimate guide on programming handheld and embedded devices.* John Wiley and Sons, Inc., USA, 2001.
14. 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.
15. N. R. Jennings. An agent-based approach for building complex software systems. *Communications ACM*, 44(4):35–41, 2001.
16. N. R. Jennings and M. Wooldridge. Applications of intelligent agents. *Agent technology: foundations, applications, and markets*, pages 3–28, 1998.
17. 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.
18. 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.

19. Y. Labrou and T. Finin. A semantics approach for kqml a general purpose communication language for software agents. In *Proceedings of International Conference on Information and Knowledge Management*, 1994.
20. Y. Labrou, T. Finin, and Y. Peng. Agent communication languages: The current landscape. *Intelligent Systems*, 14(2):45–52, 1999.
21. Z. Maamar, W. Binder, and B. Benatallah. *Agent for Ubiquitous Computing*, chapter 19, pages 395–412. Kluwer Academic Publishers, 2004.
22. Q. Mahmoud. Mobiagent: An agent-based approach to wireless information systems. In *Proceeding of the 3rd International Bi-Conference Workshop on Agent-Oriented Information Systems*, Montreal,Canada, 2001.
23. T. Rahwan, T. Rahwan, I. Rahwan, and R. Ashri. Agent-based support for mobile users using agentspeak(l). In P. Giorgini, B. Hederson-Sellers, and M. Winikoff, editors, *Agent Oriented Information Systems*, Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin, Germany, 2004.
24. 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.
25. 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.
26. M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17, 2001.
27. 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).