Design Patterns for Multiagent Systems to Elevate Pocket Device Applications.

Sameh Abdel-Naby[1] Paolo Giorgini[1] and Michael Weiss[2]

[1] University of Trento - DIT, 38100 Trento, Italy.
[2] Carleton University. Ottawa, Ontario, K1S 5B6, Canada.

Abstract. The study of Design Patterns are helping software engineers to overcome repetitive problems encounter while producing particular application or outlining specific architecture. Agents and its consistent research on multi-agent systems are increasingly playing an important role in the development of mobile applications. The implementations we focus on are related to the use of agent techniques to provide holders of pocket devices with social and location-based services. In this paper, we highlight the importance of applying design patterns concepts and theories in the development of advanced mobile service applications. We further relate our ideas to the deployment of localized rideshare system that relies on the cellular phones of interested students to facilitate their daily commutes.

1 Introduction

Software designers worldwide are reaching their objectives with less attention paid to frequent and repetitive actions made by them or by others elsewhere. Limitations on advanced technology and resources have made the process of designing a pocket-device service application containing several recurring alternatives. Design patterns [6] are helping scholars to rapidly reach their desires and accurately obtain finer results through the use of previously tested methods. In addition, using pattern descriptions - *that are outlined by particular design scheme* - has made it easy for software engineers to understand the relations between specific application components and functions.

Its autonomous and proactive behavior has made several of the efforts made in the literature of Agents [7, 19], as well as Multi-agent Systems (MAS) [8], integrate to a series of modern, innovative and promising implementations and technologies. Scholars, for example [10, 9], have attempted to resolve, describe and summarize some of the habitual mistakes that developers of MAS perform. This has made the subject of agent patterns one of the few branches coming out of the little design patterns research tree, but yet it is rapidly advancing to meet adaptive and intelligent system requirements.

Holders of pocket devices are seeking the right mobile application that performs a set of independent and delegative tasks to meet daily life necessities. Literature of Multi-agent Systems (MAS) is witnessing the success made in delivering advanced mobile services to users of pocket devices, for example Kore [12],

mySAM [13] and also MoPiDiG [20]. These applications use Agent-Oriented software engineering methodologies to build sophisticated Goal and Service-Oriented Architectures assisting the so called swifting users.

Agents programming techniques are enabling users of pocket devices to fulfill specific requirements that are hardly realized through alternatives [11]. The impact Agent-Oriented methodologies [14] left on addressing new mobile service applications requirements, such as security, context-awareness, and social-ability, made developers consider applying MAS in nowadays every mobile service implementations. A number of frameworks specialized in MAS, such as JADE [15] and JACK [16], have recently included an extra feature to enable the development of agent-based mobile applications.

In this paper, we highlight the importance of applying design patterns to the development of multi-agent systems that serve pocket devices. This will leave a great impact on enabling fast, accurate and standard future integrations and implementations. The patterns-oriented approach we secondly recommend in implementing agent-based service architecture will avoid wise developers from repeating several of their daily tasks that are required to produce sophisticated applications. It will also lead novice developers to rapidly adapt to any implementation environment.

The rest of this paper is structured as follows: Section 2 outlines our motivation. Section 3 explains the foreseen roadmap. Section 4 introduces the integration between Agent-based mobile services application and design patterns. Section 5 involves a localized rideshare system that helps students to commute from and to their university.

2 Motivation

Making vastly desired services interactive and available for mobile and computer users has been always a challenge. Many ongoing research efforts are trying to realize the optimal method to make mobile service oriented architectures more reliable and efficient. For example, a widely used technology like the mobile java client offers a standard operational model for traditional cellular phones, which has facilitated the integration between computer-based and mobile-based application development.

Agent-based systems and mobile-based applications are two promising and integrating approaches that influence Mobile-Oriented Architectures. Besides, objects-identification related technologies (e.g., wireless sensors) are helping the development of smart ambient systems that can interact with humans to improve several of their daily tasks. The merit goes to Multi-agent Systems in forming efficient, intelligent and delegative virtual communities [17]. These virtual communities are reasonably interactive, and they provide data that can be further used to deliver a specific service to particular user in a certain situation.

Developing a mobile application is both, demanding and challenging task for software designers. Design patterns theories are contributing to the development of significant tools that allow implementation practices and solutions

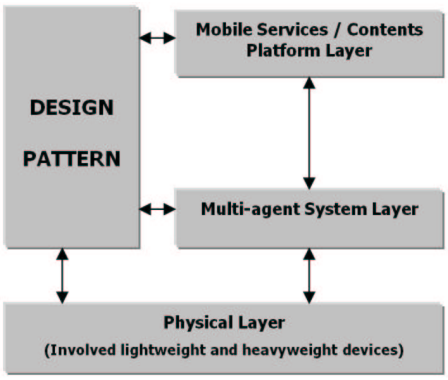

Fig. 1. The Relationships between Patterns, Mobile Services and Agents.

to be standardized and, it assist the interaction among separate system design concerns. Patterns can be seen as a directorial repository, a method to record the gained design knowledge and enable its re-use in the later development processes. Therefore, patterns are adopted to improve structural design practices. Patterns application has been carried on its precision to finally reach a specific field problem, in a new field like Mobile Services, appropriate patterns should be fulfilling the needs of mobile-application design.

The above discussion indicates the possible integration between three components, 1) mobile service application design architectures that are used to model the abstraction of delivering services to mobile-users, 2) multi-agent systems design architectures, and 3) design patterns, taking advantage of its ability to allow parts or the overall object-oriented programs to be reprocessed in a newly developed application (See Figure 1).

3 Roadmap Representation

3.1 Classification of Mobile Services

Architectures that are serving portable devices must have special characteristics in order to meet the end-users expectations. These characteristics facilitate the integration between computer-like applications and the operating systems of portable devices. Although it is not yet standardized yet it is strongly realized that modern mobile operating systems (e.g., Symbian-based mobile phones) allow mobiles to be customized to fit in users preferences.

What remains unconsidered is the standardization of several and recurring mobile-based service application components and, shaping them as plugins or separate agents that are able to interact, coordinate and finally integrate to deliver particular service.

We classify the service categories commonly used in designing modern, complex and rich pocket device application. We assume that these services are made

of combined software entities or, separate autonomous agents, each of which is in charge of achieving sub-tasks that finally help achieve the overall architectural goals, depending on what it was designed for (e.g., a communication agent, coordination agent or a negotiation agent).

These classifications are foreseen according to the feedbacks we absorbed from literature of frequently used pocket applications (e.g., applications that facilitate the delivery of information to tourists using mobile communications). Each of the main service classes is concerned about specific problem domain, and consequently, it invokes a range of sub-classes related to the abstract interest but in different solution modules.

- **Communication Class**: This class would be responsible for the exchange of data an application would try to do with other software entities to fulfill users' demands. Another function for the same class could be to ensure that a connection is established. Also the optimization of involved resources by applying a range of predefined methods. The last function of this class can be concerned with reconnecting the application with any of the collaborated actors, since some mobile applications need to launch certain offline tasks before achieving some goals.
- **Expression Class**: Depending on the type and content weight of the provided service and the running back-end architecture, each of the offered services requires certain data arrangement. Therefore, the main function of this class can control the way in which data is organized in the client-side of the architecture. Another function can be added that arranges the means of interaction between different mobile users and the displayed data. Also, a function can be added to control the displayed data in terms of whether it requires detailed or slight particular dataset execution.
- **Implementation Class**: Due to limitations on resources and because of the communication speed between end-users and operating servers, not all of the call methods of specific pocket device service is made on the client side device. The implementation class can be responsible for the organization of service tasks executed on the client side of the application (Pocket Device). The second function of this class handles the server side of the application that executes tasks involving special resources. The final function of this class is to monitor the utilized resources and fittingly organize the overloaded tasks.
- **Combination Class**: Real-life scenarios and observations may lead us to combine two or more services that would further enable mobile users to sufficiently achieve very complex tasks. For example, the combination between maps and restaurant directories would make a mobile user able to select a good restaurant and identify its location at the same time. The combination class is responsible for managing: 1) The server-side aspect that makes this class able to retrieve the required data from other linked distributed servers, 2) The application-side aspect, responsible for merging the retrieved data and make it accessible by single user interface, and 3) the client-side aspect that guides the pre-installed mobile application through the structure of collaborating servers.

3.2 Agents Service Communications

The construction of the above service classes are foreseen to be independent from the specific Agents layer they are interrelating with. However, they cannot be used without being installed in a proper agent's platform. This usually happen because they have no original means of communication and must be cross-layered with multi-agents architectures that apply a set of pre-defined communication protocols.

An example for possible integration can be with any FIPA [22] compliant framework, which uses an Agent Communication Language (ACL) to coordinate among agents. This communication language is located two layers below the mobile service application layer and on the same level as Agent Management and Agent Message Transport. It can also be extended to include Service Management modules.

Functions that can be performed by Agents to represent service classes and their sub-classes are as follows:

- The search function is used to properly retrieve a particular service class from a range of service categories previously defined by mobile-based service application developer. This function also helps the agent to recognize the service that best fit into its predefined characteristics and, it facilitates the adaptation process of a service class and the autonomous agent that together form what is called a *Service Agent*.
- The analysis function - After the agent finds its desired service class, it returns to the Agent Management architecture with a detailed description. Thus a simple analysis will be made by comparing the requirements and conditions drawn by the service with the characteristics given to an agent.
- The agent returns to the Agent Management layer, only if they are fitting, certain analysis results. Consequently, saving these results will be the third function an agent has to consider and, another sub-function is the waiting condition an agent should carry out in order to give the possibility for the overall architecture to complete its pending operations.
- The integration function collects the separate software entities and integrates / examines them for overall compatibility. The final Multi-agent system is now consisting of separate agents that are shaped as general architecture plug-ins. The loop of functions can be repetitive once plug-ins matching process produces any object failure. In order to prevent the system from replicating similar functions and producing another object failure, implementing an *Intelligent Expert Agent* mechanism is suggested.

Furthermore, the service classification modules and functions must apply and utilize some system interaction rules to ensure performance and optimum service delivery, these rules can be as follow:

1. The first rule is to partition the desired service class to as many subclasses as possible and, to represent each of them with a detailed description that reflects their exact role in the general system. That is recommended because

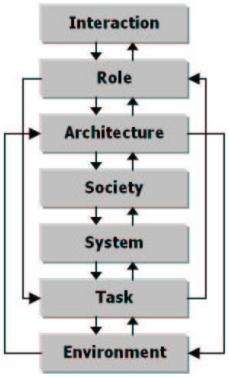

Fig. 2. Applied Design Patterns Catalog Views.

the fewer the tasks a service application is about to carry out, the easier it is to locate a proper service agent to fit in, and the clearer it will be the service expression.

2. The second rule is to link the service class to an abstract hierarchy that is found in the Directory Services part of the architecture, which will help the agent locate the appropriate classifications. The hierarchal approach is commonly used in Service Oriented Architectures (SOA) design, because it is easy to refer to as a set of services with a general title.
3. The third rule is to attach a particular service class that simulates the data flow among different parts of a specific mobile-based service application and, examine the most suitable delivery method of service contents. This takes place once the function of integrating the selected agents and examining the general coordination of the multi-agent system components is completed.

These earlier phases must include a detailed specification of the supporting technologies, which will further simplify the application of efficient design patterns. The experience obtained at each time a service class and an agent integrate will be used for further implementations.

3.3 Applied Design Patterns Catalog

The iterative engineering approach for multi-agent systems, proposed by Lind [23], has presented a new way to model and capture MAS related aspects. Lind has also presented a pattern catalog structure that we adapt in order to enable and enhance the development of agent-based service application. Modifications and interconnections were made to seven views of those presented in the catalog structure to fit in our framework (See Figure 2):

Interaction. This view presents the concept of a system that consists of multiple independent entities, which coordinate with themselves in order to achieve their individual as well as their joint goals [23]. In this view, we emphasize the need for the developer to report the methods and techniques used to interact

with the system during the design process, as well as the type of interactions made between all of the system components. Further on, these interaction experiences will be recognized in further development scenarios and arise whenever similar situations occur.

In designing a mobile-based application, certain restrictions are applied, such as limited memory and processing resources and, a unique interaction schema is required and expected to be seen. Multi-agent systems that are implemented to represent a specific environment is quite complex and may apply several uncommon interactions beside those traditional ones.

Role. According to [23], *the Role view determines the functional aggregation of the basic problem-solving capabilities according to the physical constraints of the target system.* In designing an Agent-based mobile service application, each agent is represented to solve a particular problem and accordingly, the goal of each agent is to take the predefined path to resolve a problem. In certain scenarios, more than one agent may cooperate to resolve a more complex task, whereby each of the involved agents is sub-tasked to address a specific goal. This makes the role of the agent depends on the coordination protocol used to achieve the overall mobile service delivery.

Architecture (system, agent, agent management). *The Architecture view is a projection of the target system onto the fundamental structural attributes with respect to the system design,* [23]. Repeatedly, several system integrations are made to a specific multi-agent system until predefined, or simultaneously defined, goals are achieved. These integrations are usually taking place between an existing Agent-based mobile service application and other running multi-agent systems, databases or web portals, and this is considered to enhance the quality of service provided to the end-user. In this view we outline the need to capture the full process of system integrations and, the importance of sketching the general architecture, including lightweight and heavyweight devices interrelations and roles, and make it re-usable in applying these particular types of operations on long run implementations.

Society. This view defines the structured collection of entities that pursue common goals, [23]. In a multi-agent system, separate software entities are forming virtual communities (VC) [17], and communication between these communities are not limited to single structure but, also to cross-networks architectures. Software entities of each VC are usually sharing the same interests and goals and, they collaborate to obtain certain results. In particular, if we apply the mobile service applications design approaches, emphasizing the condition of inserting a monitoring tool to observe the agents behavior in creating their own virtual communities, it will definitely increase system performance. This monitoring tool will also lead to a better prediction of society structure before implementing any of its parts and, it will prevent the system from reaching uncontrollable state.

System. This view deals with systems aspects that affect several of the other views or even the system as a whole [23]. In a pocket device service application, the System view handles the mobile user interface that relates to the interaction between the service application and the user. Covering all users' data-entry and

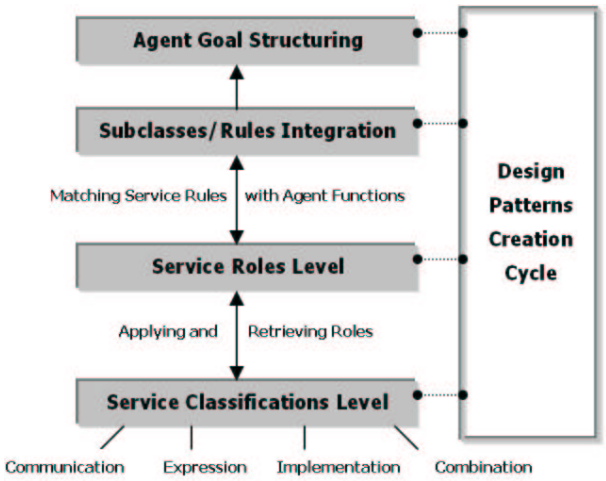

Fig. 3. General Framework Composition Phase.

portable device output functions, this view records the users' reactions in a certain situation and analyzes it to develop a better and simple service module in case a similar problem occurs.

Task. In this view, a task hierarchy is generated that is then used to determine the basic problem solving capabilities of operating entities in a running system [23]. We adopt this view as it is but we suggest a link between this view and the Role view. This link will enable the outputs, coming from the process of granting roles to agents in a mobile-based service, to be the inputs for creating a hierarchy for each agent capability.

Environment. Systems are analyzed from the developer's perspective as well as system users' perspectives. In designing an agent-based mobile service application, the developer's focus is on the way a service is implemented, tested and delivered, while for diverse systems, the focus is different. Available resources and processing algorithms control the efficiency of the overall developed architecture, regardless of how sophisticated and maintained are the systems.

3.4 General Framework Composition

In figure 3, we show the composition phase of the proposed roadmap components involves the above-mentioned four mobile service classifications and the three delegated tasks of each. In addition to its designated functions, a Service Agent will operate to be compatible with its specific service class, and a set of rules will be applied to ensure the proper match. To improve developers' design techniques, we also suggest involving Agent-oriented methodologies throughout the application of design patters Views that monitor experiences obtained while designing a certain service application.

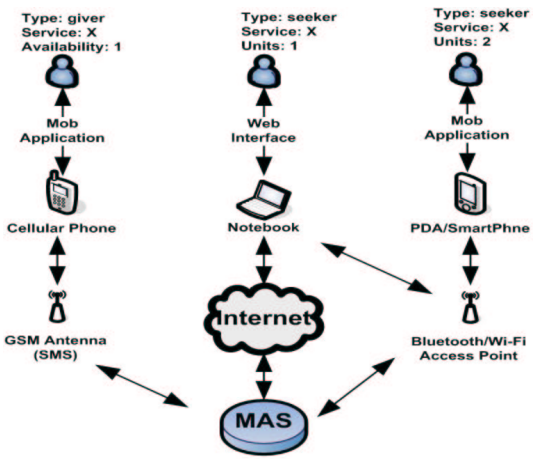

Fig. 4. Diverse communication methods establish reliable mobile service interactions.

4 Application Example

Lightweight devices are increasingly showing their necessity and reliability. Cellular phones and similar devices are part of the new telecommunications era, which made it possible to virtually carry your office anywhere you go. Nowadays, people are using pocket devices that allow them to check their emails, exchange faxes, surf the internet, edit documents and, do shopping. These services are provided through quite simple, user-friendly and well-developed interfaces, and costless with respect to the value of services users are getting.

4.1 Andiamo Rideshare Service

In figure 4, we address the motivating scenario driving our research towards the introduction of design patterns in Multi-agent Systems (MAS). We assume that three different users are interested in using the same service architecture in managing their desires to obtain a certain item. This service is limited to the demand and supply of a specific product among system users only (e.g., available care seats in a carpooling system or a used book in a trade environment). These users are using their lightweight devices to communicate with the service architecture and, each is adapted to use its device-based application. One of the users may be the giver of this product and the others are the requesters.

Each of the involved lightweight devices is configured to utilize a specific communication method to access the service, a cell phone my send service requests using SMSs, a notebook may access the service through a dedicated web interface or a Bluetooth/Wi-Fi access point. The service of interest in this example is the Andiamo rideshare system [13,21], which keeps track of available rides in the cars of participating users, and the interests of users to contribute

to the travel cost in exchange for sharing a ride. Such an interaction is usually provided by a third-party website which match service requests to offers.

Andiamo [26, 18] allows users to share car rides using their cell phones. This system helps reduce energy consumption, save money, and decrease among system users. The users use their lightweight devices to communicate with the service architecture. One user is the seller of the specific needs (e.g., destination) and preferences (e.g., sex of other user, price). When the agent joins the system, a matchmaking process takes place. An agent that carries specific information looks for another agent that may add required details to complete a task. In complex scenarios multiple agents need to cooperate to solve the task.

The matchmaking process involves two types of agents, buyer agents (BA) that act on behalf of service seekers, and seller agents (SA) that act on behalf of service providers. Each type of agent holds information related to its role in the system. A BA keeps data that helps SA increase his profit, and the data a SA keeps helps a BA achieve the overall objectives of the system. If, at the end of the matchmaking phase, a BA has found an suitable SA, both agents try to maximize their returns through a negotiation process.

As we explored this and similar application scenarios, we could observe that the same solutions recur regardless of the type of service that needs to be delivered. Examples of recurring solutions include the interaction of system components, the means of communication, and addressing security requirements.

4.2 Interaction of System Components

In general, the more software agents interact and coordinate, the more users cooperate. A negotiation model or protocol among all of the involved agents is commonly applied, so they can understand each other, discuss their desires and eventually achieve their objectives. Different negotiation protocols have been proposed by scholars, these protocols are mostly inspired by sociological, political and psychological studies about human negotiation in real-life situations such as Auctions, Peace agreements and Bidding theories. These protocols aim at facilitating the agents' mission within a systems acting in particular predefined environment, which can make it easy to standardize using design patters.

Different circumstances accompany different MAS environments, and depending on the situation a specific negotiation protocol is chosen and applied on all system agents (for example, [21]). Time, data transfer rate, general bandwidth constrains, bridge connection stability and security might not form great obstacles in computer based MAS implementations, as much as it may cause failure for mobile-based MAS application. All of the advanced negotiation protocols used in the development of agent-based mobile service applications are coming from computer/server environments or, databases and matchmaking applications.

In this paper, we claim that we should standardize on a particular agent negotiation protocol dedicated to all agent-based implementations of services delivered to lightweight devices through different communication methods (e.g., Bluetooth, or Wi-Fi). This negotiation protocol should rapidly achieve complex

tasks, because of the dynamicity of the environment and the nature of connectivity used. Therefore, when design patterns are applied to such a repetitive negotiation scenario, linking between different agents in different environments, regardless the service exchanged, will be smoothly implemented.

In the Negotiating Agents pattern [2], agents make their intentions explicit. For example, they exchange constraints on what the other agents are allowed to do. The solution involves an initiator who starts a negotiation round by declaring its intention to its peers, which are all the other agents who must be consulted before the initiator can go ahead with its action as intended. The peers take on the role of critics in the negotiation. Two roles (initiator and critic) are defined by this pattern that agents taking part in a negotiation can play.

A set of negotiation patterns for agents is described in [4]: Trusted Facilitator, Total Disclosure, and Incremental Disclosure. Each pattern deals with the problem that the agent need to align their courses of action. In the first pattern they use a trusted third party to facilitate, in the two other patterns a decentralized approach is taken. The solutions in those patterns differ in how the preferences of the agents are disclosed. In the Andiamo system, a decentralized negotiation protocol is most suitable. Using Incremental Disclosure provides buyer and seller agents with most control over the outcome of the negotiations.

4.3 Means of Communication

Multiple technologies are used to connect a wide range of mobile users in various places. Users of advanced mobile devices have the option to communicate with a location-based service using Bluetooth, Wi-Fi or internet, and with a remote service via SMS or email. No matter what service is to be delivered and the nature of the content exchanged, very similar connectivity modules are used to provide any of the involved MASs with a reliable means of communication. MAS developers and software designers, particularly those targeting mobile devices applications, are wasting time and effort by reimplementing the same functionality.

In [18], when we tried to enable our potential mobile users to communicate with the centralized MAS through Bluetooth, we had to modify, develop and integrate various parts of the desired system. And in [26], when we decided to add the capability to receive SMSs from end-users and transform them to service requests, our design was similar to that for enabling Bluetooth.

Therefore, the steps performed to enable specific MAS to establish a Bluetooth connection with a mobile device application, the procedure taken to communicate services requests through SMSs, and the way a MAS switches between those communication options could all become a standard plugin to be added as needed.

4.4 Providing Security

The importance of integrating design patterns and agent-oriented methodologies to provide better solutions for the development of secure agent-based systems

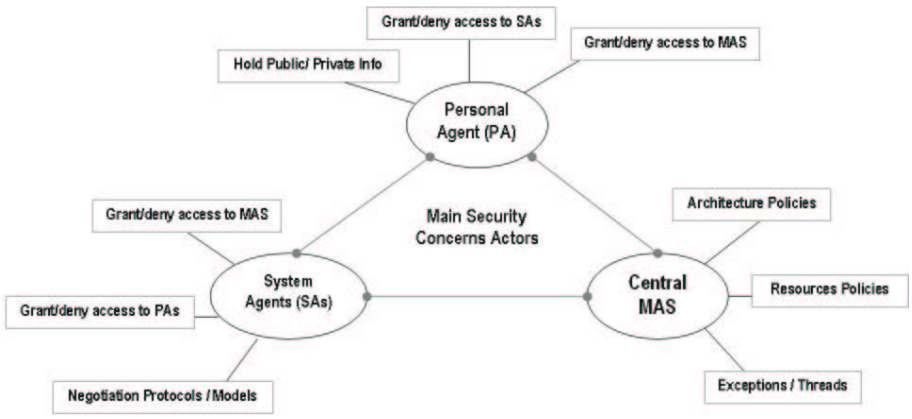

Fig. 5. Pattern-oriented approach to implement secure pocket service application..

has been identified in [3]. Agent-based mobile device applications are required to meet security criteria that exceeds those of desktop computers.

While implementing different pocket devices service applications we encountered frequent security requirements observations. We tried to capture an abstract view of these concerns and identify its parties. Therefore, in figure 5, we further apply the design patterns roadmap we proposed to these security concerns and, we try to show the usefulness these patterns would give if similar approaches were used to demonstrate negotiation and means of communications.

As shown in figure 5, in a service architecture using an agent framework to deliver and exchange contents between users of mobile devices, the major interactions are made among three different parties: Personal Agent (PA) representing the needs and preferences of a specific user; System Agents (SA) standing in for the PAs of other users with similar interests; and a Central Multi-agent System (MAS) responsible for service request matchmaking, content delivery, execution and enforcing security policies.

From PA stance, when it comes to security [28], three different functions are to be addressed: 1) the function separates between the public and private data each PA keeps (e.g., desired service price and bargaining limits). 2) Granting and preventing other system agents from accessing certain information that are of importance to the desired service to be found and, with the most proper method (e.g., the best rideshare offer with lowest price ever). 3) Granting and preventing the central MAS from collecting particular data about this specific PA and so its creator, the same for system interactivity modes and connectivity methods, which can be adjusted by users (e.g., high-level system interactions or complete system delegation depending whether the user at work or at home).

From System Agents (SAs) stance, similar to PA, three different functions are the subject of security concerns in any pocket-devices interactions environment. The two functions of granting and denying access to other PAs and the central MAS are similar to those of individual PA. The new attribute comes along with

the negotiation function, where protocols, strategies and models of cooperation are set and configured to control the interactions made among different software agents, and to ensure the construction of reliable service architecture.

From the central MAS stance, there are two functions that manage the policies previously defined by, mostly, architecture developers to control the overall and resources policies, the main algorithms used to provide the service and, the maximum or minimum amount of resources to be involved in each loop. Another function is related to the exceptions and threads this MAS is willing to perform in order to facilitate an uncommon operation (e.g., avoid user registration in peak-times).

Four patterns for secure agent systems comprising have been documented in [3]: Agency Guard, Agent Authenticator, Sandbox, and Access Controller. The Agency Guard protects an agency from malicious agents gaining access through multiple access points. Agent Authenticator requires that each agent, after they have entered the system is authenticated. If the agent cannot be authenticated it will be executed in a Sandbox. Access Controller restricts access to the agency's resources, as defined by the security policies of the agency.

5 Related Work

An approach to Agent-based service composition and its application to mobile business process was presented by scholars [24].They described an architecture model for multi-agent systems that was developed in the European project LEAP (Lightweight Extensible Agent Platform). Its main feature is *a set of generic services that are implemented independently of the agents and can be installed into the agents by the application developer in a flexible way.* These generic services are responsible of the reusability of the common software entities and they handle most of the agent-related concerns (protocol, conversation, language, ontology, and errors), while allowing the developer to concentrate on the application logic.

Another Agent System Development Method based on agent patterns was presented by other research group [25], and their method *enabled developers to design process into two architectural levels and applying the appropriate agent patterns*, and they added to the same method a higher level designs that are independent of specific agent platform so it can be reused.

6 Conclusions

The paper proposes a roadmap for using design patterns to develop Agent-based mobile service application. In this schema, we recommend a classification of mobile application services using multi-agent system development techniques. We integrated these techniques with a suggested modified version of Lind's patterns catalog, which we believe to further enable developers of pocket devices service applications to further drag and drop certain system plug-ins to enable complex

functions to be easily integrated. We finally use an example of a Rideshare / carpooling application that we developed at the University of Trento to show the usefulness of our roadmap and how it can be contributing to existing literature.

Acknowledgment

We thank the ICT laboratory of ARS LOGICA for the unabated cooperation and support given to innovative and creative ideas. We also acknowledge the partial involvements and support of these projects: EU-SERENITY, PRIN-MEnSA, PAT-MOSTRO, PAT-STAMPS, and PAT-UNIQUIQUE SUUM.

References

1. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad P. and Stal, M. Pattern Oriented Software Architecture: A System of Patterns, Wiley, 1996.
2. Deugo., D., Weiss, M., and Kendall, E., Reusable Patterns for Agent Coordination, in: Omicini, A., Coordination of Internet Agents, Springer, 2001.
3. Mouratidis, H., Weiss, M., and Giorgini, P., Modelling Secure Systems Using an Agent-Oriented Approach and Security Patterns, International Journal on Software Engineering and Knowledge Engineering, 16(3), 471-498, 2006.
4. Weiss, M., and Esfandiari, B., Patterns for Negotiating Actors, European Conference on Pattern Languages of Programs (EuroPLoP), 2005.
5. Zhao, L., Mehandjiev, N., and Macaulay, L., Agent Roles and Patterns for Supporting Dynamic Behavior of Web Service Applications, AAMAS Workshop on Web Services and Agent-Based Engineering (WSABE), 2004.
6. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional; 1st edition (January 15, 1995).
7. Stan Franklin and Art Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In the Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.
8. M. Wooldridge. An Introduction to Multiagent Systems. John Wiley and Sons, 2002.
9. Elizabeth A. Kendall, Chirag V. Pathak, P.V. Murali Krishna, and C.B. Suresh. The Layered Agent Pattern Language. In Proceedings of the PLoP'97, 1997.
10. M. J.Wooldridge and N. R. Jennings. Pitfalls of agent-oriented development. In Proceedings of the Agents-98, pages 385-391, 1998.
11. C. Carabelea and O. Boissier. Multi-agent platforms on smart devices: Dream or reality? In Proceedings of the Smart Objects Conference, France, 2003.
12. M. Bombara, D. Cali, and C. Santoro. Kore: A multi-agent system to assist museum visitors. In Proceedings of the Workshop on Objects and Agents, Pp.175-178, 2003.
13. O. Bucur, P. Beaune, and O. Boissier. Representing context in an agent architecture for context-based decision making. In Proceedings of CRR'05, Paris, France, 2005.
14. Federico Bergenti, Marie-Pierre Gleizes and Franco Zambonelli. Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook. Springer; 1 edition (June 30, 2004).
15. Fabio Luigi Bellifemine, Giovanni Caire and Dominic Greenwood. Developing Multi-Agent Systems with JADE. Wiley (April 20, 2007).

16. Michael Winikoff. JACKTM Intelligent Agents: An Industrial Strength Platform. Chapter 7 in Multi-Agent Programming, edited by Rafael H. Bordini, Mehdi Dastani, Jrgen Dix, and Amal El Fallah Seghrouchni, Springer 2005, p175-193.
17. A. Rakotonirainy, S. W. Loke, and A. Zaslavsky. Multi-agent support for open mobile virtual communities. In Proceedings of the International Conference on Articial Intelligence (IC-AI 2000) (Vol I), Las Vegas, Nevada, USA, pages 127-133, 2000.
18. Bryl, Volha and Giorgini, Paolo and Fante, Stefano. An Implemented Prototype of Bluetooth-Based Multi-Agent System. Proc. of WOA05, Camerino, Nov 2005.
19. H.S. Nwana. Software Agents: An Overview. Knowledge Engineering Review, 1996.
20. Seitz, C. and Berger, M. and Bauer, B. MoPiDiG. In: Proceedings of the First International Workshop on Mobile Peer-to-Peer Computing. (2004), Florida, USA.
21. C. Carabelea and M. Berger. Agent negotiation in ad-hoc networks. In Proceedings of the Ambient Intelligence Workshop at AAMAS'05 Conference, Utrecht, The Netherlands, pages 5 - 16, 2005.
22. The Foundation for Intelligent Physical Agents (FIPA), http://www.fipa.org, 2003.
23. Lind, J. Iterative Software Engineering for Multiagent Systems - The MASSIVE Method, volume 1994 of Lecture Notes in Computer Science. Springer, May 2001.
24. Berger, M. Bouzid, M. Buckland, M. Lee, H. Lhuillier, N. Olpp, D. Picault, J. Shepherdson, J. An Approach to Agent-Based Service Composition and Its Application to Mobile Business Processes Siemens AG, Muenchen, Germany; Mobile Computing, IEEE Transactions, Volume: 2, Issue: 3, 197- 206, July-Sept. 2003.
25. Y. Tahara, A. Ohsuga, S. Honiden. Agent System Development Method Based on Agent Patterns Proceedings of The Fourth International Symposium on Autonomous Decentralized Systems. 1999.
26. Abdel-Naby, S. Fante, S. and Giorgini, P. Auctions Negotiation for Mobile Rideshare Service. In the Proceeding of the IEEE Second International Conference on Pervasive Computing and Applications (ICPCA07), July 2007, Birmingham, UK.
27. H. Mouratidis, P. Giorgini, and M. Weiss Integrating Patterns and Agent-Oriented Methodologies to Provide Better Solutions for the Development of Secure Agent Systems. In Proceedings of the Workshop on Expressiveness of Pattern Languages 2003, at ChiliPLoP 2003, March 11-14, 2003 Carefree, Arizona.
28. Jansen, W., Karygiannis, T. (1999), Mobile Agent Security, National Institute of Standards and Technology, Special Publication 800-19