Mobile Nested Transactions Monitor based on Multi-Agent Systems: Workflow Layer

Jorge Martinez[1], Matias Alvarado[2]

[1] Artificial Intelligence Lab., CIC. National Technical University.
J. Batiz esq. O. De Mendizabal s/n., Mexico, DF, C.P. 07738
george@correo.cic.ipn.mx
[2] PIMAyC, Mexican Petroleum Institute
Eje Central Lazaro Cardenas 152. Mexico, D.F., C.P. 07730
matiasa@imp.mx

Abstract. Transaction Processing Monitors featuring Nested Transactions are in the core model for mission-critical applications. In this paper, a workflow layer for Mobile Nested Transactions is presented together with a multi-agent system implementation. The main advantage to introduce is a fault-tolerant mechanism. This is aimed to deal with both the loss of communication –very usual in mobile environments- and concurrent client access.

Introduction

The mobile computing paradigm has introduced new issues and challenges in data processing. Users are able to access their information with the help of mobile phones, personal digital assistants and portable computers. However these devices are prone to power outages, network disconnections and memory overflows. Then, control mechanisms are required in order to preserve data and information consistency. Transaction Processing is a Distributed Systems branch related to the study of data consistency. Beyond the traditional flat models –where objects are stored at the same host, the distributed approach for transaction processing encloses the scenario where there's more than one host.

The above-mentioned paradigms are distributed by definition. And so are Multi-Agent systems. Considering such shared condition, it is proposed in this paper a Transaction Processing Monitor based on a Multi-Agent system. It is also the objective of this work, to take a formal model to the ground of a real implementation. That is the case of the Logic of Interaction [1, 2], a BDI framework aimed to deal with concurrent actions interaction within a distributed agents group.

In the next section, the multi-agent and mobile computing paradigms are explained under an integrated scope. Section 3 presents a historical evolution of Transaction Processing ending at the Mobile Nested approach, for which a Monitor is under development. Section 4 describes the workflow layer for the Transaction Processing Monitor and a control mechanism for transactional behavior in agents. Conclusions and undergoing work are addressed in Section 5.

Multi-agent framework for mobile computing

Multi-agent systems have arisen as an alternative approach for solving distributed problems. In fact, these kinds of systems are derived from research efforts in the field Distributed Artificial Intelligence. It was the development of message exchange mechanisms among smart-like computer systems, which pushed the idea for modelling each entity as an agent.

The basic features in the expected behaviour of an agent are: autonomy, that implies that the agent (to some extend) keeps control over its internal state and the way it behaves over an environment; social ability, the agent must be capable of interact and communicate with other entities during the problem-solving process; and learning, that is, evolve through its life-cycle and acquire both new knowledge and abilities. Additional features of agent and multi-agent systems include those related to reasoning, mobility and persistence. It is also expected that agents perform sensing operations over their environment as well as adaptability to deal with unexpected events. Agents are designed to reach their goals under conditions described for their information (knowledge/beliefs) [21].

Advances in wireless networking and portable information applications have introduced a new paradigm known as mobile computing. Users are no longer restricted to working at fixed sites. With mobile devices, such like phones, personal digital assistants and laptop computers, users can access their information despite of their geographical location. The multi-agent approach has been firmly related to dealing with problems that are distributed in nature and located at fast evolving environments. One example for these kinds of problems can be found in the field of so called mobile computing [21]. A mobile computing environment is shown in Figure 1; it is basically composed of: Fixed Hosts, Mobiles Hosts, Mobile Support Stations. These latter provide wireless network access to a limited scope range named Cell.

A Mobile User is that one who gets connected to the fixed network through his/her mobile host. It would be fairly desirable to maintain a connection alive while the user is on the road. The problems with a mobile environment arise when the device loses its connection by any reason. The most frequent are low batteries, physical location of the device (e.g. the user enters subway station) and links instability. From these issues, it becomes feasible to implement multi-agent systems over mobile groups, that is, one or more agents per device, which interact in order to fulfil a set of goals. The main advantage to achieve is the development of a fault-tolerant mechanism that deals with the lost of communication that may occur between two devices in a mobile environment. The underlying issues of combining these two technologies have been treated in [9] and [15]. This latter also mentions a mobile database approach like the one introduced later in the present work. Examples of multi-agent systems over mobile groups have been implemented for securing an electronic marketplace [10] and as schedulers for travellers [8].

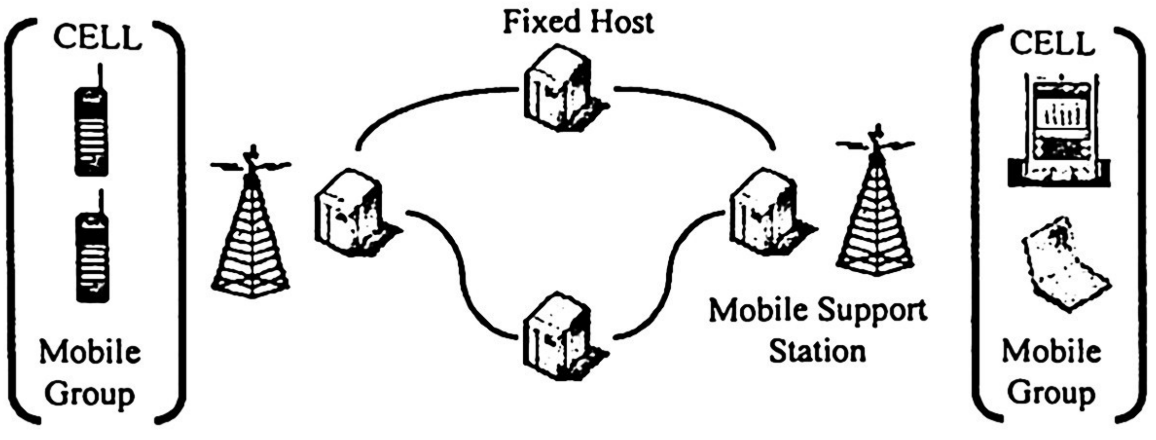

Fig. 1. Mobile Computing environment featuring two service Cells

3 Mobile nested transactions

The concept of Transaction can found in almost any type of business process currently modelled by object-oriented techniques. In [5], it is paid special attention to Transaction Processing as a hallmark for the next Client/Server technology generation.

A definition from [7] mentions that a transaction works as a mechanism for preserving consistency in the set of working objects. A common belief about the origin of transactions, establishes that they come from Database Management Systems theory. They were born from the user requirement for executing a set of operations over a database as a unit. The transaction idea was introduced in the theory of Distributed Systems under the form of Transactional File Servers [16]. There, the objective was to provide file access service to concurrent users over a network. File consistency required protection against multiple requests for both reading and writing.

In the present work, the goal is not only to control database access, but also to coordinate a mobile group that involves logical units and physical devices.

3.1 ACID properties

Transaction models are deeply based on the preservation of four basic properties. *Atomicity* stands for "all or none" and implies that the set of operations comprising a transaction must be executed as a whole. Although *consistency* is considered as a transaction property, it actually refers to data state from the database; however it is the responsibility for a transaction to take the database from a consistent state and to leave it in a similar one. *Isolation* means that a transaction must think of itself as the only one in execution at a given moment. *Durability* stands for data persistence; once a transaction has been committed, changes over database objects cannot be undone but by the execution of a second transaction.

Safekeeping the ACID properties, is the ultimate task to be achieved in transaction processing. However, recent works [11] in the concurrency control side propose the relaxation of isolation conditions. The benefit is an improvement in the number of concurrent clients; they are allowed to access a shared resource without introducing further conflicts in the read/write operations.

3.2 Concurrency control and distributed transactions

A client is responsible of requesting the execution of operations to a server. The natural consequence of multiple clients requests, arises as a concurrency control issue over the objects to be used. It depends on whether a transaction needs to read or write an object that the server blocks the resource for shared or exclusive access respectively.

A flat transaction can be thought of a set of atomic operations over a group of database objects. These operations are organized under a partial order [14]. Three bracket operations mark the context of a transaction: *Begin, Commit, Abort*. These latter are invoked at the end in order to indicate whether the transaction execution succeeded or not.

Every request belonging to a flat transaction is executed over objects from the same server. In a distributed transaction, there are two or more servers holding the resources that will be read or modified by a transaction. Thus, there is necessary to have a Coordinator in charge of managing the success or failure result from the other participants. This is commonly achieved under the Two-Phase Commit Protocol (2PC) [19].

3.3 Nested transactions

The idea of Nested Transactions extends the distributed concept by allowing other transactions (known as children) to be born under the context of a parent transaction [7, 17, 18]. In this way, there will be a tree of nodes executing read/write operations and capable of spawning inner sub-transactions. Available computing resources are the unique limitation for the tree deepness; however, it is no longer required to complete all the operations in one place. Each child could be executed in different locations, that is, nested transactions are distributed. In [12, 13], it has been concluded that nested transactions feature special conditions under which it is not possible to comply with the four ACID properties. Except for the root node, children and leaf nodes fail ensuring *Durability*. The reason is that any changes over the database are actually performed until the root node commits or aborts. A transaction tree example is depicted in Figure 2.

Nested Transactions suit in mobile applications involving wireless or cellular connections for the following reasons. A compound transaction with nested children, offers better performance in concurrency and fault tolerance. Children could be executed in a parallel fashion and each one holds local responsibility for committing or aborting. However, none of the operations are actually reflected in the database content until the root transaction commits or aborts. Some other Nested Transactions features include the optionally of success for children nodes. According to this, a root transaction is able to commit even if some of its branches fail during execution.

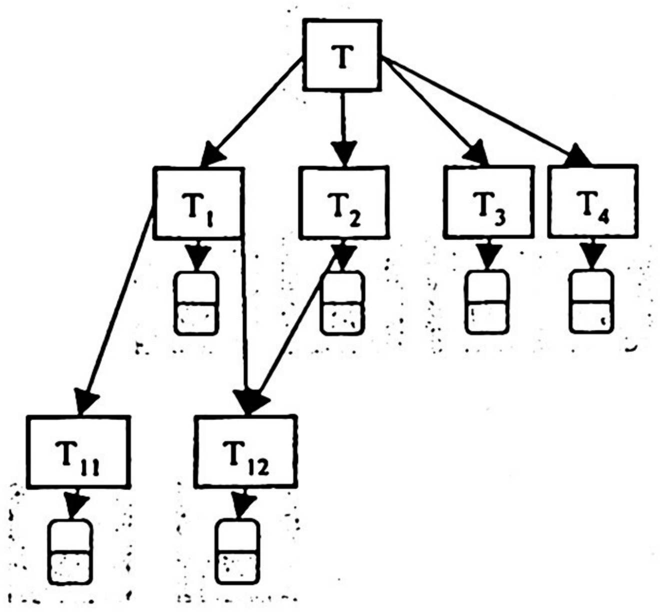

Fig. 2 Nested transaction. The Root node T spawns four children. T_1 spawns two more children resulting in four leaf nodes. It can be seen that T_{12} actually works as sub-transaction for both T_1 and T_2. T_3 and T_4 work at the same node

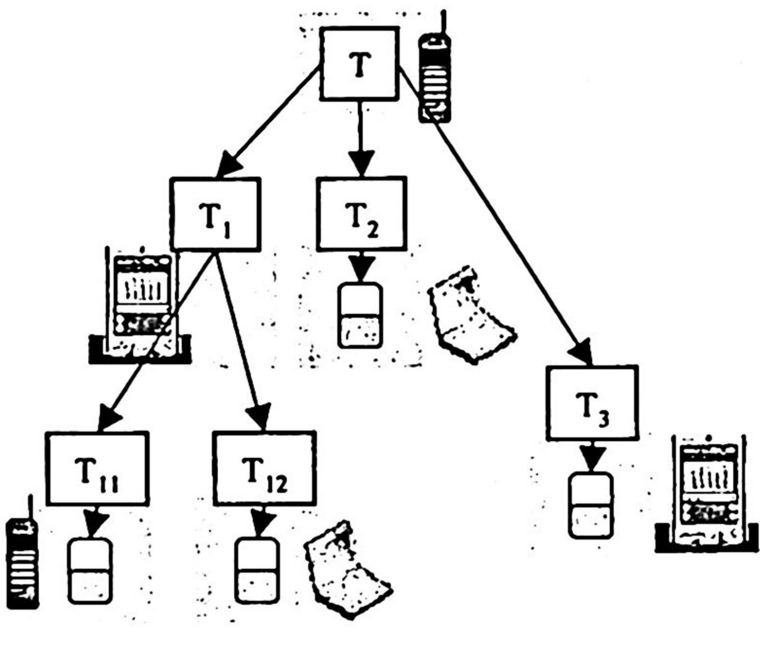

Fig. 3. Mobile Nested Transaction

3.4 Mobile nested transactions

In [12] the concept of *Mobile Nested Transactions* is introduced. These are the result of combining the concepts of Nested Transactions with Mobile Computing, particularly, a group of mobile hosts. There, a mobile device represents each node in the transaction tree. A noticeable difference between the general model and the above proposal is found in the read/write access. In [17], only leaf nodes are able to access database objects whereas the [14] definition enables intermediate nodes to execute such operations as well. In contrast with the general definition from 3.3, the next remarks are considered in the Mobile Nested approach: child nodes have only one parent, read/write operations are actually done at the lowest (leaf) level and, each transaction node is located at one mobile host. This can be seen in Figure 3.

4 Mobile nested transactions monitor

4.1 Underlying formal model

Logic of Interaction [1], [2] was introduced as a formal model for multi-agent systems, aimed to deal with the balance agent's knowledge and actions. The central issue is that individual agent actions do interact. So, an action representation must make these interactions explicit and need to explicitly model different processes. The important aspect is the way in which agents perform the actions. In this sense, provides a model for synchronized, parallel, sequential and concurrent actions carried out by a single agent and by a group. Modal temporal logic formalism is used for action representation and modelling.

The key elements from the Logic of Interaction are related to Beliefs (pre-conditions), Goals (post-conditions) and Action execution itself. There has been defined a set of operators for interactions control. These are aimed to help in differentiating actions based on their space and time nature. Based on this, agents can perform: sequential, parallel, synchronized and concurrent actions. The full definitions concerning the Logic of Interaction are out of the scope of this paper.

In order to pave the way that connects a formal model with the actual implementation, only some elements from the logic of interaction have been translated to code. The system has been implemented in an open environment using JADE (Java Agent DEvelopment Framework) as the agent platform [6]. Agents use ontologies in order to represent internally pieces of information [3]. This is used in the exchanged messages and for inner control and operations.

4.2 Workflow layer

A Transaction Processing Monitor is conceived as the software in charge of managing simple requests from users that will be scaled over a distributed system [4]. It is also responsible for safekeeping the ACID properties during concurrent transactions execution. In a few words the monitor: receives some request, translates it into a system understandable language, triggers the transaction beginning, controls the commit or abort operations and finally, reports the result to the user.

According to [5], Transaction Processing Monitors featuring Nested Transactions as the core model for mission-critical applications will better reflect the business process nature. Three layers define the gross structure of Transaction Processing monitor:

Presentation. It receives instructions from the user and translates them into a system understandable language. These instructions are sent to the workflow layer. The result of the transaction is later presented to the user.

Workflow. As the name implies, this layer is responsible for routing, managing and answering the requests received from the presentation layer. Instructions are turned to the third layer and results are sent back to the first layer.

Database. The actual access to database objects is achieved at this level. The results, either successful or not, are informed to the previous layer. Currently, the implementation features an embedded database at each device: Pointbase Micro Edition [20]. Since the agents development framework is Java-based, it was required a compatible database tool that could run over mobile devices.

At this point, it has been implemented a basic workflow layer for the Mobile Nested Transactions Monitor. In particular, it has been implemented a Nested Two-Phase Commit. In this mechanism, a set of participants join the transaction started by some user at a root node. Descendant nodes may join each participant until a tree is completed. Once the tree is ready, each leaf informs its parent whether they failed or succeeded in executing their assigned set of operations. Eventually, all this information reaches the root node, then:

Phase 1. The root node asks all the successful nodes to get ready for commit. Since intermediate nodes may abort locally, only those descendants that are sons of successful nodes receive the *canCommit* request sent by the root node.

Phase 2. With the retrieved answers, the root node decides to commit or abort and informs the final decision to all of the involved nodes in the 2PC.

The above process is shown in Figure 4.

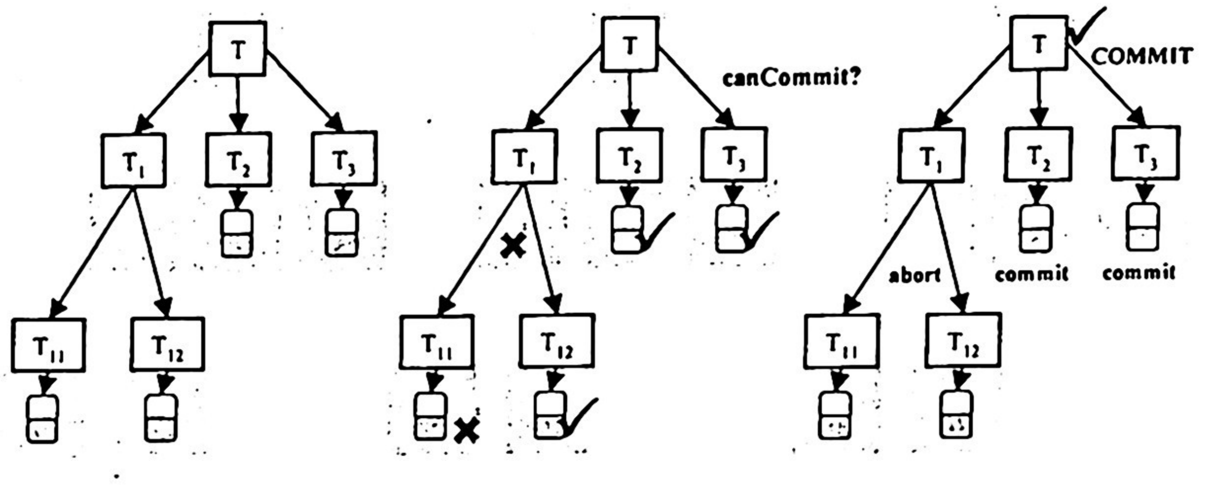

Fig. 4. Twb-Phase Commit protocol in a Nested Transaction. Since T_{11} and T_{12} do not receive a final commit or abort, they abort by default after a timeout

4.3 Control mechanism for transactional agents behaviour

During transaction processing, agents may find that some data objects are already in use due to concurrent access by other clients. In order to address this sort of events, a control mechanism has been proposed. This is an embedded behavior in each agent taking part in a given transaction. Agent's actions are classified into mandatory and optional, strong and weak; the combination of these traits and the expected behavior can be found in Table 1.

Table 1. Actions classiffication supporting concurreny control in agents interaction and nested transactions processing. The description column shows what to do in case of action failure

Type	Sub-Type	Description
Mandatory	Strong	The transaction is aborted.
Mandatory	Weak	A new attempt is completed unless the user cancels.
Optional	Strong	It is ignored for this transaction but another transaction is scheduled for later attempts.
Optional	Weak	No more attempts are done.

The above classification helps an agent-node in the transaction tree in decision-making. This may happen while trying to access a data object that is already locked, or even when communication is lost with one of the agent's siblings. This is achieved by introducing alternative ways for the node to operate with a different set of descendants from the original one. In this way, an agent is not fully required to wait for an object to get unlocked. If the failed operation is not mandatory, a new attempt can be done, either in the same transaction or by scheduling a future transaction. There is introduced a notion of action *persistence* in the sense that the agent, will attempt to complete their assigned set of instructions –whenever it is possible. Previous user feedback is required in order to set the configuration parameters that will rule the agent's behavior for this mechanism.

As host crashes and breaks in communications are expected events in a mobile environment, the proposed mechanism works as a base for logging and recovery controls. On the other hand, better concurrency control is achieved through a Lock Manager [11, 7]. These two mechanisms are part of an undergoing effort and currently out of the scope of this paper.

5 Conclusions

The multi-agent systems approach is firmly devoted to problems that are distributed in nature and located at fast evolving environments, such as those found in the so called mobile computing. Under such context, users are able to access and process information with the help of mobile devices. However, these latter are prone to operation outages that may endanger data consistency.

The Mobile Nested Transaction concept was introduced as an alternative to deal with the ACID safekeeping challenge in the operation of a set of mobile hosts. There, sub-transactions are allowed to born inside the context of a parent transaction. Children nodes failures no longer imply that the root parent has to abort the complete transaction. A Transaction Processing Monitor is under development; it is based in a multi-agent system. In this paper, it was presented the Workflow layer featuring a Mobile Two-Phase Commit protocol.

The Logic of Interaction is used to rule the agent's action coordination. It is implemented in an open environment (JADE). A control mechanism was introduced in order to achieve a transactional-like behaviour in the agent's participation within a Mobile Nested Transaction.

Pending work is heavily related to implement the other two layers from the monitor: Presentation and Database access. In this latter, a Lock Manager mechanism is also considered. It will help to improve concurrency control in shared data objects.

References

1. Alvarado, M., Sheremetov, L.: Modal Structure for Agents Interaction Based on Concurrent Actions. In V. Maik, J. Müller, M. Pchouek (eds.): Multi-Agent Systems and Applications III: 3rd International Central and Eastern European Conference on Multi-Agent Systems. Lecture Notes in Computer Science, Vol. 2691. Springer-Verlag, Berlin Heidelberg New York (2003) 29-39
2. Alvarado, M., Sheremetov, L., German, E., Alva, E.: Logic of Interaction for Multiagent Systems. In: C.A. Coello Coello, A. de Albornoz, L.E. Sucar, O.C. Battistutti (eds.): MICAI 2002: Advances in Artificial Intelligence: Second Mexican International Conference on Artificial Intelligence. Lecture Notes in Computer Science, Vol. 2313. Springer-Verlag, Berlin Heidelberg New York (2002) 378-396
3. Bellifemine, F., Poggi, A., Rimassi, G.: JADE: A FIPA-Compliant agent framework, Proc. Practical Applications of Intelligent Agents and Multi-Agents, April (1999), 97-108.
4. Bernstein, P.A., Newcomer, E.: Principles of Transaction Processing. Morgan Kaufmann Publishers Inc. (1997)
5. Byte Archive at http://www.byte.com/art/9504/sec11/art1.htm
6. Caire, G.: JADE Tutorial: Application-defined content languages and ontologies. TILab S.p.A. (2002)
7. Coulouris G., Dollimore J. , Kindberg T.: Distributed Systems. Addison Wesley (2002)
8. van Eijk, R.J., Ebben, P.W.G., Bargh, M.S.: Implementation of a scheduler agent system for traveling users. In proc. of Workshop on Ubiquitous Agents on embedded, wearable, and mobile devices. Bologna (2002)
9. Finin, T., Joshi, A., Kagal, L., Ratsimor, O., Avancha, S., Korolev, V., Chen, H., Perich, F., Cost., S.: Intelligent Agents for Mobile and Embedded Devices. International Journal of Cooperative Information Systems (2002)
10. Fischer, K., Hutter, D., Klush, M., Stephan, W.: Towards secure mobile multiagent based electronic marketplace systems. Electronic Notes in Theoretical Computer Science. Vol. 63. Elsevier Science (2002)
11. Gama, L.A., Alvarado, M.: Concurrency control for Read-Only in Mobile Nested Transactions. In proc. of the 2nd Workshop on Intelligent Computing in the Petroleum Industry ICPI (2003)
12. Gama, L.A., Alvarado, M.: Mobile Nested Transactions for Nomadic Teams. In: Alvarado, M., Sheremetov, L., Cantu, F.: Special Issue on Intelligent Computing for Pretoleum Industry. Elsevier. (2003)
13. Gama, L.A., Alvarado, M.: Transacciones para Cómputo Móvil: presente y perspectiva futura. Revista Digital Universitaria, Vol. 3. No. 4. http://www.revista.unam.mx (2002)
14. Gray, J., Reuter A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers, Inc. (1993)
15. Loke, S.W.: Supporting Intelligent BDI Agents on Resource-Limited Mobile Devices - Issues and Challenges from a Mobile Database Perspective. In proc. of Workshop on Ubiquitous Agents on embedded, wearable, and mobile devices. Bologna (2002)
16. Mitchell, J.G., Dion J.: A Comparison of two network-based file servers. Comms. ACM, Vol. 25, No. 4. (1982) 233-45

17. Moss, J. E. B.: Nested Transactions: An Approach to Reliable Distributed Computing. MIT Press, Cambridge, MA (1985)
18. Nested Trans.at http://www.cs.panam.edu/~meng/Course/CS6334/Note/master/node89.html
19. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems. 2nd Ed., Prentice-Hall, Inc.(1999) 381-401.
20. Pointbase Micro Developer's Guide at http://www.pointbase.com/support/docs/pbmicro.pdf
21. Wooldridge, M.: An Introduction to Multi-Agent Systems. John Wiley & Sons. England (2001)
21. Zaslavsky, A., Tahir Z: Mobile Computing: Overview and Current Status. Australian Computer Journal. Vol. 30. No. 2 (1998)