ARTÍCULO

Concurrency Control Monitor for Nested Transactions based on Autonomous Agents

Jorge Martinez[1], **Matias Alvarado**[2*]

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

Abstract

Transaction processing monitors featuring nested transactions are in the core model for mission-critical applications. In this paper, an atomic commit protocol for nested transactions over mobile devices combined with a concurrency control mechanism is introduced. The monitor features a flexible and goal persistent technique dealing with the loss of communication – a latent risk in mobile environments–, whilst safekeeping the transaction ACID properties. The implementation runs over the JADE framework. The initial results of performance for complex enough tests on a mobile devices network are presented.

Keywords: Concurrency Control, Nested Transactions, MultiAgent-Based Monitor.

1. Introduction

The mobile computing, understood as data-objects management over wireless devices, 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.

In distributed systems the transaction processing (TP) concerns to the study of data consistency. A TP system must handle concurrent requests over data-objects. It is expected an efficient handling that avoids errors such as inconsistent or lost results at reading/writing data items [Bernstein97]. Beyond the traditional flat models -where all the data-objects are stored at the same host and the operations are serialized, the distributed approach for transaction processing encloses the scenario where there's more than one host. So, the transaction's operations can be executed in different hosts or nodes and the data-objects stay through the nodes as well.

Powerful cases of distributed transactions are nested transactions (NT). The NT concept extends the distributed one by allowing other transactions, known as children to be born under the context of a parent transaction [Coulouris02].

The mobile computing as well as the NT paradigm they are distributed by definition. And so are Multi-

[*] *Corresponding author.*

Agent Systems (MAS). Considering such shared condition, it is proposed in this paper a MAS-based Transaction Processing Monitor; a hybrid technique taking advantages from both approaches as detailed in a later section. Nodes involved in transaction processing are subject to intense message interchange. Software agents provide that functionality as well as: autonomous execution, service-oriented interaction and goal-oriented behaviour. The JADE (Java agent development) agents provide the required coordination mechanisms for distributed and nested transaction processing. Heterogeneity on hardware architectures, operating systems and DBMS (DATA Base Management System) is also supported.

In Section 2, the multi-agent as well as the mobile computing paradigms are explained in an integrated standpoint. Then the Mobile Nested Transaction approach follows in the next section, whereas the agent-based monitor structure including the atomic commit protocol, is described in Section 4. The core contributions of this work, an action classification table in addition to the control rules for lock management, being the mechanisms dealing with concurrency control are explained in Section 5. The experiments and results reports occupy the following section, and discussion, in progress work and conclusions are addressed in the final section.

2. MAS for Mobile Computing

Multi-agent systems have arisen as a powerful approach for solving distributed problems. The basic features in the expected behaviour of an agent are: autonomy, social ability, and learning. This latter conveys agent's adaptability and evolution during its life cycle. Additional features of agent and MAS 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.

Advances in wireless networking and portable information applications have detached users from their workstations at fixed sites. With mobile devices, 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. These kinds of problems are found in the field of so called mobile computing [Zaslavsky98]. A typical mobile computing environment is shown in Figure 1. It is basically composed of:

Fixed Hosts. These ones are resource servers under the traditional networking concepts. Usually, all of them are attached to a fixed facility. This is mainly due to the heaviness of processes and processes number running concurrently in a given moment. A subset of Fixed Hosts is called Mobile Support Stations (MSS). These hold the responsibility of providing services of wireless interfaces to Mobile Hosts. Their scope range (limited to a geographical radius) is known as Cell. Both Fixed Hosts and Mobile Support Stations are connected via a fixed public or private network.

Mobile Hosts. This group comprises any type of device capable of establishing a wireless (or cellular) link to a Mobile Support Station. It is this way that the device gets connected to a fixed network. MSSs are responsible for all of the hosts found inside the scope range; nonetheless, whenever a host leaves this range, another station must provide the connection service.

Problems within a mobile environment arise when some device loses its connection due to: low batteries, change of the physical location and links instability. Our purpose is the development of a more persistent and fault-tolerant mechanism, that deals with the loss of communication between devices in a mobile environment likewise in [Finin02]. In [Loke02], there is a mobile database approach like the one introduced in the present work. In [Vogler98], there is an implementation for distributed transactions; this one focuses on agent mobility and relies on the CORBA Object Transaction Service for coordination of the global transaction.

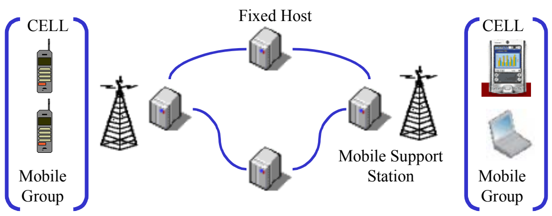

Figure 1. Mobile Computing environment featuring two service Cells

In [Nagi01], the robustness of action execution by MASs is addressed by an agent architecture based on the open-nested transactions model. In [Straßer98], there is another transaction-oriented

MAS: the purpose is to build an open system as an enabling technology for the evolution of an electronic marketplace. In that system, the mobility issue is addressed by providing migrating mechanisms. It also features a strong differentiation between service agents and user agents. Examples of MAS over mobile groups have been implemented for securing an electronic marketplace [Fischer02] and as schedulers for travellers [vanEijk02].

3. Mobile Nested Transactions

3.1 Transactions Basics

The concept of Transaction is found in the business world where enterprise - individual exchanges take place, and the result of exchanges is recorded. So, there are programs performing administrative functions over shared data [Bernstein97]. In computing terms a transaction works as a mechanism for preserving consistency in the set of working objects [Coulouris02], namely, operating system, databases and data warehouses as well as software deployments. Moreover, the transactional deployment of any information system module is currently an obliged standard [IBMWepSphere04], [Intervowen04], [Microsoft04], [Tallman95]. 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.

Recent works in the concurrency control side propose the relaxation of isolation conditions [Kempster99], [Seifert03], [Gama03]. The benefit is an improvement in the efficiency of concurrent processes as they are incremented simultaneously. They are allowed to access a shared resource without introducing further conflicts in the read/write operations.

As above mentioned the nested transactions model advances the distributed one by allowing the children transactions born for a parent transaction [Gama03b]. In this way, there will be a tree of nodes executing read/write operations and capable of spawning inner sub-transactions. Each child could be executed in different locations. The nested transactions model regards a special condition with the ACID properties compliance: 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 [Gama03b]. A transaction tree example is depicted in Figure 2.a.

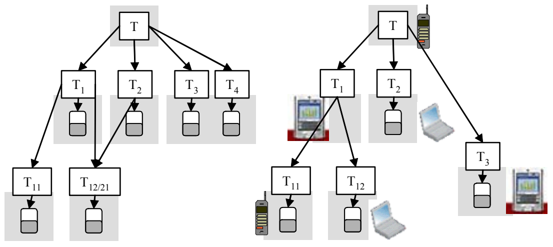

Figure 2. (a) Nested Transaction. (b). Mobile Nested Transaction

3.2 Flexibility of the Nested Approach

Nested transactions are apposite for offering better performance in concurrency and fault-tolerance. For instance, in a compound transaction with nested children, these can be executed in a parallel fashion as 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.

Moreover, NTs are suitable for mobile applications involving wireless or cellular connections. In [Gama03b] the concept of *Mobile Nested Transactions* is the result of combining Nested Transactions with Mobile Computing, particularly, a group of mobile hosts. There, a mobile device can represent a node in the transaction tree. A noticeable difference between the general model and the above proposal is found in the read/write access. In the classical work on NT [Moss85], only leaf nodes are able to access database objects whereas the [Gray93] definition enables intermediate nodes to execute such operations as well. Our mobile nested approach resembles the first definition in the sense that read/write operations are actually done at the lowest (leaf) level; moreover, the following remarks are observed: child nodes have only one parent and each transaction node could be located at one mobile host or share its device with another node, see Figure 2.b.

The present project bases nested transactions processing on a MAS monitor. From the group of agents' point-of-view, there come up two key issues: a goal persistency concern and a control concurrency problem in the case of multiple requests to access a data object. Transactions completion must be kept under real-time constraints of the system.

4 The Transaction Processing Monitor

A Transaction Processing Monitor is the software in charge of managing requests from users that will be scaled over a distributed system [Bernstein97]. It is also responsible for safekeeping the ACID properties during concurrent transactions execution. Thus, the monitor:

- Receives some request,
- Translates it into a system understandable language,
- Triggers the transaction beginning,
- Controls the commit or abort operations, and
- Reports the result to the user.

Currently, the monitor implementation is based on the JADE framework and has been tested with the Pointbase Micro Edition [Pointbase04] software. This is a Java-based rDBMS that goes embedded in the mobile device.

Three layers roughly define the structure of a Transaction Processing monitor [Bernstein97]: *Presentation*, *Workflow* and *Database*. Each agent must be capable of providing any service related to the three layers. For instance, a user may start some transaction on its own device, the node may be working as a workflow manager for a set of operations or it could be executing data-access commands on the internal Pointbase tables. Agents also must be light enough in order to fit the computing power of a mobile device. In the current architecture, the structure of every agent working as part of the monitor is basically the same. However, during transaction processing each agent assumes a different role and if required, it is enabled to undertake a second role for a concurrent transaction.

Agents are designed to behave as: root (presentation), the one that receives input from the user interface, translates the instructions to a system understandable language, sends them to the next layer, retrieves the responses and shows the result to the user; child (workflow), under this role, the agent takes a set of instructions from the upper level, routes them to its own children nodes, makes decision on committing or aborting the branch under its scope and sends back the response; leaf (database), at this level, the agent applies the reading/writing operations over its own embedded data tables and reports the result to its parent. This is illustrated in Figure 3. There, agent A operates as workflow manager for sub-transaction S_1 and at the same time, it serves as the root node for transaction T. As for agent B, it works as leaf node (the one that actually executes operations over database objects) for S_{121} while also manages workflow for sub-transaction T_1. Agent C controls leafs T_3 and S_{122} for their respective roots T and S_1.

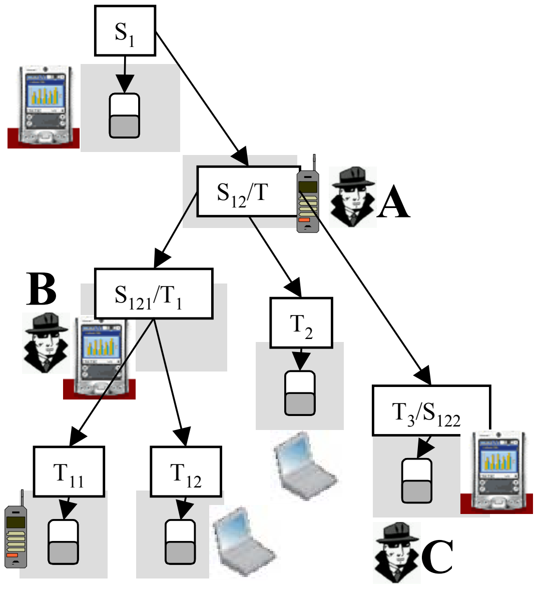

Figure 3. Agents playing simultaneous roles

4.1 Advantages of the autonomous agents support

Each agent provides access to data stored locally and runs without strict dependency to (relative autonomy) a central system, turning process monitoring into a combination of local/distributed tasks. It is local since every agent makes decisions over their internal state and it is distributed for constant communication is required among the group of agents thus establishing emergent interactions that affect the transaction's global result. Otherwise, relying on a specialized node implies a greater degree of vulnerability for data management. The following traits are also noticed with an agent approach: greater transactions concurrency, isolation level relaxation, commitment/abortion independence of each transaction output, message interchange supported by a codification language and semantic information in the message content. These advantages increase the odds of improved throughput of the application.

The resulting transaction tree follows a hierarchical scheme; nevertheless, each agent remains autonomous from the decision-making perspective. There is no centralized entity that coordinates the actions from the group of agents. Agent's actions that respond to their environment are heavily related to their current state and persistent goals. Besides, agents can participate as nodes from concurrent transactions in a peer-to-peer interaction either with its parent or its children nodes.

MAS positively stress concurrency transaction since each agent locally executes pieces of work in a given node. Agent autonomy reduces dependency between transactions thus avoiding locked-object latency and reducing deadlocks and cascade-aborts. Deadlocks are found when some node enters an idle state while waiting for the release of a busy data-object that has been locked by another node. In turn, this latter, in order to keep on working, is also waiting for a third node to release its locks, and so on until a closed cycle is formed by every node waiting for a neighbour to release some blocked objects. It has been calculated the little probability for deadlocks under a reasonable concurrent transactions number [Bernstein97].

Agents supporting Nested Transactions reach their goals in spite of other transactions or siblings' failures. This is due to the timeout mechanism and action classification agents use to react after they find blocked objects in their way.

Data distribution in heterogeneous DBMS and processing distribution using Autonomous Agents are a robust alternative for environments characterized by instability and frequent failures. A group of wireless devices is distributed by definition and operated by autonomous users. Besides, they feature mobility as a basic property. Mobile agents represent an excellent framework for NT implementation on mobile devices since they are able to support off-line operation in the following ways.

- Unfinished tasks can be transferred to an available device or can be resumed by the original device proved that this latter is back on-line.
- Agents communicate effectively while keeping a decentralized control of the global transaction status and tasks assigned to each agent.
- They could process a transaction locally while staying disconnected from the wireless network.
- Programmers are taken from the rigid client/server model to a more flexible peer-to-peer model, in which process communicate as peers and act either as clients or servers.
- Software agents also endure applications scalability as those take processes to the data location, rather than bringing data to the place where it is required for processing. That is an ideal feature for NT processing with mobile devices [Gray96].

The multi-agent architecture is heavily based on the Logic of Interaction [Alvarado03], thus taking formal mechanisms to the application ground. The advantage is a reinforcement of the system robustness with the underlying theory as the wire-frame structure for the multi-agent interaction.

Following, we explain how the monitoring controls (embedded at each agent) operate during transaction processing.

4.2 Atomic Commit Protocol

At this point, it has been implemented a Nested Two-Phase Commit. With this mechanism, a set of participant agents join the transaction (started by some user at a root node) and receive a subset of operations from the full transaction. Descendant nodes may join each participant until a tree is completed. Once the tree is ready:

Phase 1. A *canCommit* request is sent by the root node to its children nodes. This request is hierarchically forwarded downwards and eventually reaches the leaf-level. From here, nodes report either YES or NO upwards until the information reaches the root node. Intermediate nodes make a provisional decision according to those reports received from their children.

Phase 2. With the retrieved answers, the root node decides either to commit with the successful branches or to abort the global transaction. This is informed in the multi-level fashion as in Phase 1. Children with an aborted ancestor must abort as well.

The above workflow process is shown in Figure 4. There, the root node decides to commit despite of T_1's failure. Since T_{12} is a child of T_1, it has to abort as well.

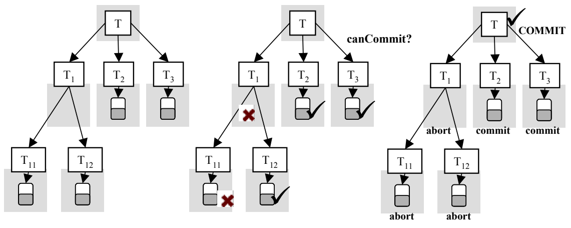

Figure 4. Two-Phase Commit protocol in a Nested Transaction

5. Concurrency Control mechanism

During transaction processing, agents can find that some data objects are in use by concurrent clients. To address this sort of events, a control mechanism being an embedded behaviour in each agent taking part in a transaction, is proposed. Transaction and sub-transaction nodes are classified into Critical, Mandatory Strong, Mandatory Weak and Optional. Table 1 shows what to do in case of failure at transaction execution on the first attempt.

Type	Description
Critical	The transaction is aborted.
Mandatory Strong	More attempts are scheduled until the transaction succeeds or a timeout expires and the transaction is aborted.
Mandatory Weak	The transaction is sent back to the user for further instructions.
Optional	More attempts are scheduled until the transaction succeeds or a timeout expires and the transaction is sent back to the user as in the above case.

Table 1. Classification of nodes for concurrency control.

In addition, the following rules apply:

- If a Critical/Mandatory Strong transaction fails, the whole branch is unsuccessful. That is, all of the nodes at the same level belonging to the same parent are aborted as well.
- If for a given node, all of the children are classified as Mandatory Weak or Optional, it is only necessary the success of one child in order to take the whole branch as successful.

The above classification helps an agent-node in the transaction tree during the decision-making process. 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.

As host crashes and breaks in communications are expected events in a mobile environment, the proposed mechanism also works as a base for logging and recovery controls.

5.1 Locks-based concurrency control

The control mechanism that we introduce is aimed to deal with conflicts that arise when two or more nodes (either from the same transaction or from different transaction) try to access the same data object. **It is the objective then, for this locks-based concurrency control to implement simple yet effective methods**. These are tied to operations from agents and in conjunction, will globally emerge as a smart-like collective behaviour during transaction processing. Considering the conditions that endanger device operation in mobile computing, the concurrency control must be designed to manage shared resources under time constraints. This mechanism itself helps in dealing with possible disconnections, power outages and storage overflows.

Our proposed mechanism is illustrated with the help of Figure 5. There, sub-transaction T_{13} from T_1 has already locked a data object and a second sub-transaction T_{211} from T_2 tries to access the same object.

First able, it is worth recalling that there is an agent running at each mobile device. This agent has an embedded micro Lock Manager that handles a table with information on locks and data objects at the host device. Locks are held during the lifetime of a transaction and they are immediately released after a final commit or abort is received from the Root Node.

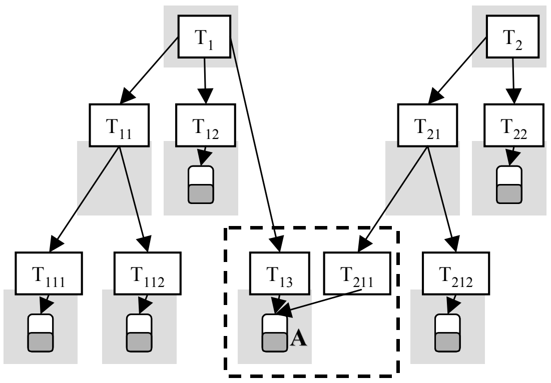

Figure 5. Concurrent access illustration: T_{13} is already using A when T_{211} tries to access the same object. The embedded Lock Manager follows some simple rules to conflict solving.

Suppose that there is a shared-lock request for reading access from T_{13}, the Manager registers which transaction and object are engaged during the time it takes until resolution. If a second transaction, T_{211}, is willing to read the same object, the Manager is able to grant shared-access level since there are no conflicts among requests.

Problems arise when at least one of the transactions needs exclusive-access level, as it is willing to write over the data object. Under such conditions, we introduce the next rules in order to deal with conflicting access requests.

5.2 Examples

To illustrate the above mentioned scenario, let's suppose that in Figure 4, the agent coordinating node T_{13} has already locked A for exclusive access. Then, if a second transaction T_{211} arrives with the intention to use A, it will receive a *wait* notification so then T_{211} enters an idle state.

While T_{211} is on standby, two events can happen: T_{13} releases the object and T_{211} can continue its assigned tasks until it completes; else, T_{21} reaches the timeout after which it must verify that its child T_{211} is ready or not so it can forward the information to the upper level. T_{211} receives the request from T_{21}, it has no choice but to answer *No* and it will abort. T_{21} must then decide whether it is possible to commit the sub-transaction. This decision is made upon the control mechanism introduced in 5.1 According to the protocol from 4.1, T_2 will eventually be aware that the branch failed or succeeded. If the compromised node (T_{211}) unleashed a global abortion due to its priority, then the agent managing T_2's Root Node may schedule a new attempt to accomplish the unfinished tasks. User input before transaction execution is particularly helpful since he/she may specify whether the retrial is automatic or there will be need for user authorization to do so.

A deadlock resolution mechanism is aimed to deal with scenarios where some transaction enters a wait state due to a locked object by a second transaction. This latter is also waiting for another object held by a third transaction and so on until a cycle is formed. At this point, any transaction is not able to continue since they become part of a deadlock ring. Therefore, some action must be undertaken in order to break the cycle. Deadlock resolution by means of node chasing is part of undergoing work.

6 Experiments and results

6.1 Test description

Ongoing experiments are conducted with the Java 2 Micro Edition toolkit that provides a PC-based emulator for a generic mobile phone, a PalmOS-based Tunsgten C with MIDP4PALM, and transactions-nodes running both on desktop and laptop computers. A total of 7 devices (1 simulated, 1 real and 5 PC-agents) supporting 4 concurrent transactions, each having each shape in Figure 6.

The aim is to test the system's performance with different configurations on number of intermediate nodes and leafs. The tree deepness varies from the simplest flat distribution (Fig. 6.a) to the ladder configuration (Fig. 6.d). The monitor can support additional levels. Each model executes writing operations at the leaf level, except the tree shape experiment (Fig. 6.c), where the central nodes run read-only operations. The four transactions are launched concurrently.

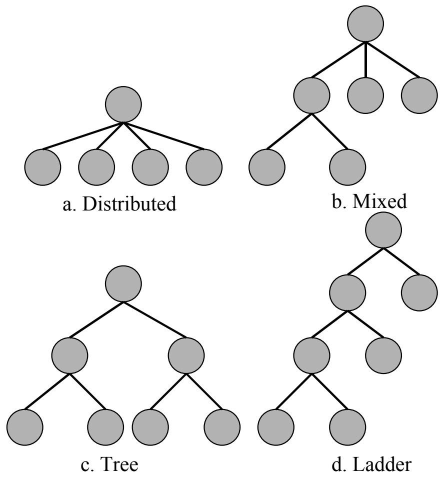

Figure 6. Experimental transactions models

Each device can play the role of root, participant or leaf. Within the above-mentioned conditions, each device plays an average of four simulateneous roles. On the other hand, a node may open up to four concurrent connections to its embedded database. Now, it is opportune to recall the storage and computing limitations of mobile devices, especially those of the Wireless Toolkit Emulator and the Tungsten handheld.

The embedded database holds a single table with four different columns, each with a defined data-type: fixed character array, integer number, date value and number with decimals. Although the Java 2 Micro Platform does not support floating point, Pointbase provides a DECIMAL format for simulating this type of data.

Each node randomly selects one action from Table 1 for each transaction or sub transaction that is being monitored: either, mandatory or optional sub transactions and their effect over the global transaction.

6.2 Results

Preliminary results show a satisfactory performance in the successful/failed transactions tradeoff, considering the relatively slight concurrent requests. About 60% transactions do commit in the distributed and mixed cases, whereas for the ladder and tree experiments, 75% transactions do. The results are deeply related to each sub transaction classification: there are cases such that a single branch is enough for committing the global transaction. This is mainly found in the ladder and the tree-shape transactions.

In general, 75% of the leaf-nodes succeeded. It is noticed that children nodes playing 4 or 5 roles, reported the lowest success rate to their parents, only around the 33% of transactions.

The computing power of the Tungsten device impacts the nested transaction performance. In our experiments the Tungsten played as the root node for the distributed example, and as an intermediate node in the ladder example. It takes 20512 milliseconds as average processing time for the ladder configuration, and 22562 for the distributed case. In contrast, it takes an average time of 7528 and 4670 milliseconds to complete the tree and mixed transactions respectively.

In addition, the Tungsten cannot hold more than two simultaneous roles due to the restrictions of the MIDP Virtual Machine for Palm OS -64 KB as the maximum for heap memory. In contrast, the emulated mobile phone (provided with the Sun Wireless Toolkit) does well with up to 5 simultaneous roles. A similar stress capability on a real device is expected on scheduled experiments having more intense concurrency.

7 Discussions

The MAS approach is heavily favoured by the agents' autonomy and social abilities. As well, the cooperative nature of the agents interactions enables the distribution of a large assignment into smaller tasks. Tasks execution and coordination is attended by the whole group -not by a centralized entity.

In [Vogler98] there is an implementation for distributed transactions focusing on agent mobility that relies on the CORBA Object Transaction Service for coordination of the global transaction. In the present paper we introduce an implementation based on an Agent development kit (JADE), such that the main goal is to define a base layout for further and more complex mechanisms.

In [Nagi01] the robustness of action execution is addressed by a MAS architecture based on the open-nested transactions model. In contrast, our work addresses the closed-nested transaction counterpart. In spite of the mobile devices popularization among users, workgroups of this kind often comprise a relatively small number of participants. This is reinforced by the delegation of specific tasks, like setting a group appointment or sharing user's data with the rest of the group. Hence we expect a slight number of concurrent requests over shared resources; the control mechanism is designed on a preventive basis, a popular approach in larger applications [Lewis02].

A transaction-oriented MAS for building an open system as enabling technology for the evolution of an electronic marketplace is in [Straßer98]. There, the mobility issue is addressed by providing migrating mechanisms. There is also a strong differentiation between service agents and user agents. In our approach, the agent that interacts with the user can execute the instructions itself instead of requesting the service from a third provider.

Our experiments have shown off mobile devices capabilities. Task distribution proved to be a useful advantage for overcoming the restrictions of small devices. The ladder and tree tests perform better than the distributed and mixed counterparts, according to their success rates. The reason is that load charge, on the first ones, is better distributed over the mobile group. However, it concerns on the technical issues as more powerful devices are daily introduced to the market.

At some extent we can say that for scaling transaction loads requires more powerful devices like laptop computers. In additional experiments, we are running 16 concurrent transactions using only PC-agents. It is noticeable that success rates are similar to those mentioned above. Nevertheless, the transaction processing time is highly contrasted as it takes an average of 700-900 miliseconds (versus 4 - 20 seconds in experiments with small devices) to complete each transaction –either successfully or not.

Conclusions

A nested transaction model implemented over a group of mobile devices is introduced. A MAS embracing an atomic commit protocol and a concurrency control mechanism monitors the transactions processing. These embedded controls in the agents activate according to the role each agent plays for a transaction. The mobile two-phase commit protocol collects sub-transaction results being fed to the root node, in order to make the final decision concerning the transaction outcome.

The lock manager –at the monitor workflow layer– improves the concurrency control in shared access over data-objects. The concurrency control conflicts are solved by a proposed classification of agents' actions, deciding whether unsuccessful operations should be rescheduled for a new attempt or not.

Currently, database tests are performed through JDBC connections to the embedded Pointbase tables at each of the leaf nodes. Experiments include heterogeneous software and hardware platforms, namely PC-based emulators and real devices connected through a wireless LAN.

With an average of 68% of committed transactions, the monitor fits likely for small mobile workgroups. As mentioned in the Discussion, there is a need for role balancing in order to reach satisfactory concurrency levels without overflowing the PDA or phone capabilities. As far as experiments show, scaling the amount of transactions relies on more powerful devices that withstand the increased load.

References

[Alvarado03] M. Alvarado, M., L. Sheremetov. Modal Structure for Agents Interaction Based on Concurrent Actions. In V. Marîk, J. Müller, M. Pchouek (eds.): CEEMAS-2003. Lecture Notes in Computer Science, 2691. pp. 29-39. Springer-Verlag, Berlin Heidelberg New York (2003).

[Bellifemine97] F. Bellifemine, A. Poggi, G. Rimassi. JADE: A FIPA-Compliant agent framework, Proc. Practical Applications of Intelligent Agents and Multi-Agents, pp. 97-108. April (1999).

[Bernstein97] P. A. Bernstein, E. Newcomer. Principles of Transaction Processing. Morgan Kaufmann Publishers Inc. (1997).

[Coulouris02] G. Coulouris, J. Dollimore, T. Kindberg. Distributed Systems. Addison Wesley (2002).

[vanEijk02] R. J. van Eijk, P. W. G. Ebben, M. S. Bargh. Implementation of a scheduler agent system for traveling users. In Finin, T., Perich F. (eds): Proc. of Workshop on Ubiquitous Agents on embedded, wearable, and mobile devices. AAMAS-02 (2002).

[Finin02] T. Finin, A. Joshi, L. Kagal, O. Ratsimor, S. Avancha, V. Korolev, H. Chen, F. Perich, S. Cost. Intelligent Agents for Mobile and Embedded Devices. In Klusch, M., Zambonelli, F. (eds.): International Journal of Cooperative Information Systems, 11-3,4. pp. 205-230. World Scientific Publishing Company (2002).

[Fischer02] K. Fischer, D. Hutter, M. Klusch, W. Stephan: Towards secure mobile multiagent based electronic marketplace systems. Electronic Notes in Theoretical Computer Science 63. Elsevier Science (2002).

[Gama03] L. A. Gama, M. Alvarado. Concurrency control for Read-Only in Mobile Nested Transactions. In Braunschweig, B., Alvarado, M., Bañares-Alcantara, R., Sheremetov, L. (eds.): Proc. of ICPI'03, associated to IJCAI'03. pp. 89-93 (2003). Available from: http://www.imp.mx/icpi/docs/ICPI03%20Workshop.pdf.

[Gama04] L. A. Gama, M. Alvarado. Mobile Nested Transactions for Nomadic Teams. In: Alvarado, M., Sheremetov, L., Cantu, F.: Special Issue on Intelligent Computing for Petroleum Industry. *Expert Systems with Applications* 26-4. pp. 105-113. Elsevier Science (2004).

[Gray93] J. Gray, A. Reuter. Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers, Inc. (1993).

[Gray96] R. Gray, D. Kotz, S. Nog, D. Rus, G. Cybenko. Mobile agents for mobile computing. Technical Report PCS-TR96-285, Dept. of Computer Science, Dartmouth College (1996) Available from: ftp://ftp.cs.dartmouth.edu/TR/TR96-285.pdf

[IBMWebSphere04] Information Center for WebSphere Application Server. Available from: http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/tjta_entra2.html; INTERNET.

[Intervowen04] Intervowen OpenDeploy 5.0: Ease-of-use in Multi-Tier Distribution of Code and Content. Available from: http://www.interwoven.com/news/press/2001/0419odpr.html; INTERNET.

[Kempster99] T. Kempster, C. Stirling, P. Thanisch. Diluting ACID. ACM Sigmod Record, 28-4. pp. 17-23. ACM Press (1999).

[Lewis02] P.M. Lewis, A. Bernstein, M. Kifer. Database and Transaction Processing: An Application-Oriented Approach. Addison Wesley Publishing Company (2002).

[Loke02] S. W. Loke. Supporting Intelligent BDI Agents on Resource-Limited Mobile Devices - Issues and Challenges from a Mobile Database Perspective. In Finin, T., Perich F. (eds): Proc. of Workshop on Ubiquitous Agents on embedded, wearable, and mobile devices. AAMAS-02 (2002).

[Microsoft04] Microsoft TechNet: Deploying Content on a Commerce Server 2000 Site. Available from: http://www.microsoft.com/technet/prodtechnol/comm/comm2000/deploy/csdepcon.mspx; INTERNET.

[Moss85] J. E. B. Moss. Nested Transactions: An Approach to Reliable Distributed Computing. MIT Press, Cambridge, MA (1985).

[Nagi01] Nagi, K. Transactional Agents: Towards a Robust Multi-Agent System (ed). Lecture Notes in Computer Science, 2249. Springer-Verlag, Berlin Heidelberg New York (2001).

[Pointbase04] Pointbase Micro Developer's Guide at http://www.pointbase.com/support/docs/pbmicro.pdf; INTERNET.

[Seifert03] A. Seifert, M. H. Scoll. Processing read-only transactions in hybrid data delivery environments with consistency and concurrency guarantees. Mobile Networks and Applications, 8-4. pp. 327-342. Kluwer Academic Publishers (2003).

[Straßer98] M. Straßer, J. Baumann, F. Hohl, M. Schwehm. ATOMAS: A Transaction-oriented Open Multi Agent-System. Final Report. Institute for Parallel and Distributed High Performance Systems, University of Stuttgart (1998).

[Tallman95] O. H. Tallman. Project Gabriel: Automated Software Deployment in a Large Commercial Network. Digital Technical Journal, 7-2. pp. 56-70. Digital Equipment Corporation (1995).

[Vogler98] H. Vogler, A. Buchmann. Using multiple mobile agents for distributed transactions. In proc. of 3rd IFCIS International Conference on Cooperative Information Systems. pp. 114-121. New York (1998).

[Zaslavsky00] A. Zaslavsky, Z. Tahir. Mobile Computing: Overview and Current Status. Australian Computer Journal. 30-2. pp. 42-52. Australian Computer Society (1998).