A Knowledge Transaction Processing Language and Model for Mobile Application

J. Chen[1,2] and Y. Zhang[2]

[1] IBM Global Service Australia
20 Berry Street, North Sydney, NSW 2060, Australia
E-mail:jchen@au1.ibm.com
[2] School of Computing and Information Technology
University of Western Sydney
Penrith South DC, NSW 1797, Australia
E-mail: yan@cit.uws.edu.au

Abstract. Abstract. A new environment model is proposed in this paper to study transaction processing, intelligent agent and knowledge base in mobile environments. This model combines the features of mobile environments and intelligent agents. Another very innovative aspect of this paper is that a knowledge transaction processing language and model for mobile environments is defined and formalized by using logical programming as a mathematic tool and formal specification method. We use this knowledge transaction language and model to study transaction processing in mobile environments.

1 Introduction

Studying intelligent agent and knowledge base in mobile environments is a very new and meaningful research area. A manager uses mobile host to do the rule based decision-making and negotiation is a very practical study case in this research area. The issue of data and knowledge transactions has presented new challenges for researchers in mobile environments because of the features of mobile environments. There seems to be a separation between multiagent systems and the intelligent agents community on one side, and the mobile agents community on the other side [12]. As so far very few work has been done and no any formal study has been conducted to the issue of knowledge transaction in mobile environments, as the first step, this paper proposes a knowledge transaction processing language and model in mobile environments. The paper is organized as follows. In section 2, we give some background knowledge on intelligent agents and mobile environments, and then introduce our new environmental model. In section 3, we describe the transaction processing in mobile environments and introduce relevant concepts of logic programming. In section 4, we formalize a knowledge transaction processing language and model in mobile environments. In section 5, we give a transaction example to demonstrate how a knowledge transaction is processed based on our transaction language and model. In section 6, we conclude and summarize our work.

2 Environment Model

When we study the transaction processing [3, 6] in mobile environments, we choice the following environment model to represent the salient features of mobile environments discussed in the paper [2, 7, 9]. We are looking at a Home Server (HS) acting as permanent storage of Mobile hosts' (MH) Files. There are Mobile Support Stations (MSS) providing services to a MH when it is within its cell. The MSS is connected to the HS via hardwires. The MH is continuously connected to a MSS via a radio link while accessing data. It may, however, become disconnected either voluntarily or involuntarily. There is a centralized database residing in the HS. On each mobile host, MH, there resides a transaction manager which preprocesses transaction operations; a scheduler which controls the relative order in which transaction operations are executed; a recovery manager which is responsible for commitment and abortion management and a cache manager. The transaction study model in the mobile environment is then described as shown in Fig.1. When we study the intelligent agent in

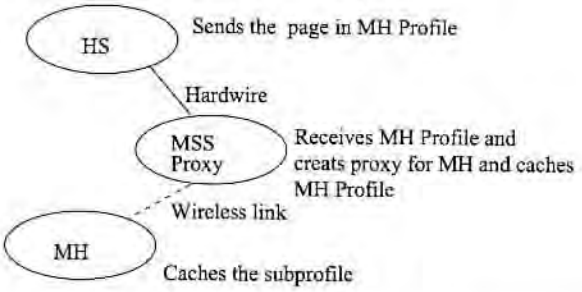

Fig. 1. Transitions Study Model in mobile environments.

non-mobile environments, we usually use the following environment model [10] as shown in Fig. 2.

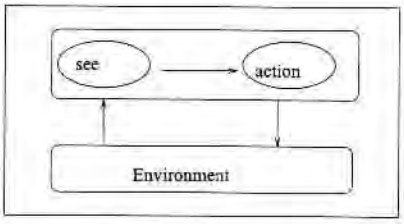

Fig. 2. Intelligent Agent Model.

The idea is that function see captures the agent's ability to observe its environment whereas function action represents the agent's decision making process. Fundamentally, an agent is an active object with the ability to perceive, reason and act. We assume that an agent has explicitly represented knowledge and a mechanism for operating on or drawing inferences from its knowledge. We also assume that an agent has the ability to communicate. In a distributed computing system, agents communicate in order to achieve better goals of themselves or of the system in which they exist. In order to study intelligent agent and knowledge base in mobile environments, we propose a new environment model by integrating the features of mobile environments and intelligent agents as shown in Fig. 3.

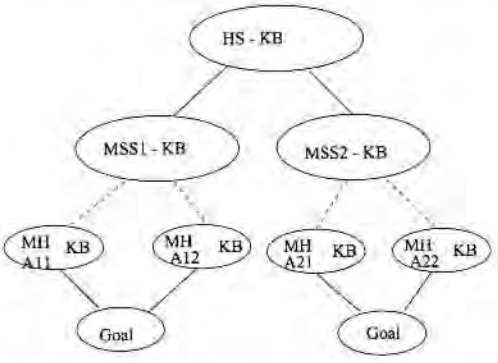

Fig. 3. Intelligent Agent in mobile environment.

In the environment model above, we assume that every Mobile Host (MH) has its own knowledge base (KB) and intelligent agent (A11, A12, A21, A22), every MSS has knowledge base residing on it as well, MSS1 and MSS2 represent different MSS in different geographic areas. In Home Server (HS) level, there is a knowledge base which has a set of rules in it. Every intelligent agent on MH is

3 Mobile Transaction Processing and Logic Programs

A typical transaction T inmobile environments will look like [4]: Contactproxy $\rightarrow r[x] \rightarrow w[x] \rightarrow inv \rightarrow$ Contactproxy $\rightarrow s \rightarrow wy \rightarrow \cdots \rightarrow c$
contactproxy $\rightarrow \cdots \rightarrow$ Contactproxy $\rightarrow s \rightarrow wu \rightarrow \cdots \rightarrow c$
in which the proxy first acquires the appropriate set of locks for the MH. Once a write is executed, the proxy is asked to broadcast a report of invalidation to proclaim the existence of a new version of x. In this example, the MH, after serving a number of operations of the T, decides to go to sleep voluntarily. This is done by first contacting the proxy and then flushing all its dirty pages (to the MSS) and releasing all the write-locks it holds. Upon waking up, the MH

goes through a process similar to start-up, i.e. there will be fetches of data objects (possibly through the proxy) and [9] requests through the proxy. The remaining operations are then executed under the covering of the locks. Finally, the transaction is committed through delayed writes [13] to the MSS. In the case of involuntary sleep the proxy is not contacted and completed with MH in another one. In this case, the transaction T becomes:
Contactproxy $\rightarrow r[x] \rightarrow w[x] \rightarrow inv \rightarrow move \rightarrow$ Contactproxy $\rightarrow \cdots \rightarrow c$
Here we have lumped all the activities related to handoff and the creation of a new proxy by a new MSS into the operation move.

3.1 Extended Logic Programs

In non-mobile environments, traditional logic programming is used as a knowledge representation tool. An important limitation of this method is that logic programming does not allow us to deal directly with incomplete information, and therefore we only can get either yes or no answer from a query. This is because in the traditional logic programming, closed world assumption is automatically applied to all predicates [5], understand that there is a major different between the scenario that the transaction fails and the transaction hangs on due to mobile user's sleep. Therefore, in mobile environments, we need a method which can deal with incomplete information directly, and this method should handle the transaction fails in the sense of its negation succeeds and transaction does not succeed in the mobile situation. The extended logic programs [5] can overcome limitation above from traditional logic programs, it contains classical negation $\neg$ in addition to negation-as-failure not. An extended logic program can include explicit negative information. In the language of extended programs, we can distinguish between a query which fails in the sense that it does not succeed and a query which fails in the stronger sense that its negation succeeds. By adding mobile semantics to extend logic programs, we can use this method to study transaction processing and deal with the incomplete information directly in mobile environments. In mobile semantics, classical negation $\neg$ is defined as explicit negative information, is explicit no when transaction is explicit fail. In the situation the mobile host is in voluntary or involuntary sleep, and therefore the information is incompleteness, we say it is absent of atom A, noted by $notA$, therefore it is unkown. We consider closed world assumption from mobile semantics as following; If mobile host is in voluntary or involuntary sleep and sleep time is beyond the limited time, we say we assume $\neg p$ is derived from not p at this time, unkown become no for the transaction. We can express closed world assumption by the rule $\neg p(x) \leftarrow notp(x), x/geT$. Here T is the time limit for mobile hosts' sleep. An *extended logic program* is a set of rules of the form

$$L_0 \leftarrow L_1, \cdots, L_m notL_{m+1}, \cdots, L_n,$$

Where each L_i is a literal.

Now we give a Yale Shooting Problem (YSP) example [1] to demonstrate how to represent knowledge in logic programs. The syntax of the language contains

variables of three sorts: situation variables S, fluent variables F, and action variable A. Its only situation constant is S_0, and $res(A, S)$ denotes the new situation that is reached after the action A is executed in situation S. The atom $holds(F, s)$ means that the fluent F is true in situation S. Usually, to represent predicates and functions whose value changes over time, term fluent is introduced. In the Yale Shooting Problem, there are two fluents: *alive* and *loaded*, and three actions: *wait*, *load*, and *shoot*. We know that the execution of loading leads to the gun being loaded, and that if the gun is shot while it is loaded, a turkey Fred dies. We have following rules for this example:
$Y1 : holds(F, res(A, S)) \leftarrow holds(F, S), notab(A, F, S)$,
$Y2 : holds(loaded, res(load, S)) \leftarrow$,
$Y3 : ab(shoot, alive, S) \leftarrow holds(loaded, S)$,
$Y4 : holds(F, res(A, S)) \leftarrow ab(A, F, S)$.
Let S_0 be the initial state, and suppose we are given that
$Y5 : holds(alive, S0) \leftarrow$.
It is easy to see that program *gamma* entails $holds(alive, res(load, S_0))$ and $\neg holds(alive, res(shoot, res(wait, res(load, S_0))))$.

4 A Logic Programming Based Transaction Language and Model

In this section, we start with a complete transaction stage by stage to give readers a clear idea what activities are supposed to happen on MH, MSS and HS at every stage. Then, we define a logic programming [5] based knowledge transaction processing language (TPL) and impose all necessary rules to formalize a knowledge transaction model in mobile environments.

4.1 Transaction Processing Activities

Firstly, let us see what activities happen on MH, MSS and HS on every transaction stage [9]. Here, we specify the activities for transaction start and commit as well.

Startup: Startup is the initial powering up of the mobile computer. When the MH is powered on, it registers with MSS. The MH notifies the MSS of its HS address. MSS then creates the proxy process which retrieves the MH profile from HS. HS sends the pages in the MH profile. The proxy receives and caches the MH profile.

Start Transaction: The MH requests a query or update transaction. The MH acquires a lock if it is an update transaction. The MSS submits the transaction request to HS. The HS does the transaction after MSS submits the transaction request.

Sleep: There are essentially two types of sleep voluntary and involuntary. Voluntary sleep is a planned power down, while involuntary sleep is an unplanned power down, i.e., the system crash or run out of battery power. There are different activities during voluntary and involuntary sleep.

Wakeup: Wakeup is the powering up after a MH has been asleep. Although similar to startup, wakeup has slightly different semantics. The wakeup sequence is as follows: Upon wakeup, the MH waits to get MSS's address (beacon). Upon receiving the beacon, the MH sends wakeup notification to MSS and requests missed messages.

Move/Handoff: The MH listens for MSS beacon. When the MH notices that it is in a different region, it contacts the new MSS. The new MSS contacts the old MSS to get the state of the MH proxy. The old MSS flushes any dirty pages to HS and sends the proxy state to the new MSS. The new MSS proxy contacts HS to tell it where to contact the MH. The new MSS proxy broadcasts any invalidations to MH.

Commit Transaction: The HS commits or aborts transaction according to the two phase commit protocol. The HS sends the transaction result to MSS. The MSS broadcasts the transaction result to MH. The MH updates the local knowledge base according to the transaction result.

4.2 Formalizing a Knowledge Transaction Processing Language

We define our Transaction Processing Language (TPL) to formalize transaction related action and fluent functions at MH, MSS and HS level. We start by introducing basic sort of functions to characterize the basic components of our language. We use actions and fluents to denote transaction processing activities, results and status. We use x denotes MH, y denotes MSS, y1 denotes MSS1, y2 denotes MSS2, and z denotes HS. At each level of MH, MSS and HS, we define some actions and fluents. The arguments and values of them will be clear from their use in the rules below in section 4.3.

Mobile Host (MH) level: We have the following actions: $Move(y, x)$, $Query(x)$, $Write(x)$, $Acquire-lock(x)$, $Flush(x)$, $Release-lock(x)$, $Request-message(x)$, $Fetch-message(x)$, $Update-knowledge(x)$. We have the following fluents: $Registered(y1, x)$, $Query-requested(x)$, $Trans-start(x)$, $Update-requested(x)$, $Locked(x)$, $Vol-slept(x)$, $Sleep-sig(x)$, $Invol-slept(x)$, $Lock-cancelled(x)$, $Update-lost(x)$, $Wakeup-sig(x)$, $Message-received(x)$, $Registered(y2, x)$, $Knowledge-updated(x)$, $Commit-broadcasting(y, x)$, $Abort-broadcasting(y, x)$, $Timeout1(x)$. Action examples: $Move(y, x)$ denotes MH moves into MSS cell, $Query(x)$ denotes MH has a query transaction request. Fluent examples: $Registered(y1, x)$ denotes MH has registered in MSS1, $Query-requested(x)$ denotes MH has requested to start a query transaction.

Mobile Support Station (MSS) level: We have the following actions: $Create-proxy(y1, x)$, $Create-proxy(y2, x)$, $Retrieve(y)$, $Cache(y)$, $Broadcast(y)$, $Mark(y)$, $Submit(y)$, $Update-sleeptime(y)$, $Buffer(y)$, $Page-broadcast(y1, x)$, $Page-broadcast(y2, x)$, $Cancel-lock(y)$, $Flush(y1)$, $Broadcast(y, x)$. We have the following fluents: $Proxy(y1, x)$, $Proxy(y2, x)$, $Cached(y)$, $Broadcasting(y)$, $Marked(y)$, $Trans-submitted(y)$, $Buffered(y)$, $Page-broadcasting(y)$, $Timeout(x)$, $Move-sig(y2)$, $Flushed(y1)$, $Commit-broadcasting(y, x)$, $Abort-broadcasting(y, x)$, $Commit-noticed(z, y)$, $Abort-noticed(z, y)$.

Home Server (HS) level: We have the following actions: $Send-page(z,y)$, $Mark-MH(z,y)$, $Do-trans(z)$, $Kill-proxy(z,y)$, $Send-dirtypage(z,y2)$, $Commit(z)$, $Notice-commit(z,y)$, $Notice-abort(z,y)$. We have the following fluents: $Sent(z,y)$, $MH-marked(z,y)$, $Trans-started(z)$, $Proxy-killed(z,y)$, $Trans-committed(z)$, $Commit-agreed(z)$, $Abort-agreed(z)$, $Timeout2(z)$, $Commit-noticed(z,y)$, $Abort-noticed(z,y)$.

4.3 Formalizing a Knowledge Transaction Model

Based on the transaction processing language defined above, we start to specify and impose all necessary rules to formalize a logic programming based transaction model in mobile environments, modeling all transaction activities, requests, results and constraints on MH, MSS, and HS three levels. In our language, only situation constant is S_0, and $res(A,S)$ denotes the new situation that is reached after the action A is executed in situation S. The atom $holds(F,s)$ means that the fluent F is true in situation S.

Mobile Host (MH) level:

Register: When MH moves into MSS cell, it is registered. The rule for this is
$r1 : holds(registered(y1,x), res(move(y1,x),s)) \leftarrow$

Start a query or update transaction: For a query transaction, as long as MH has a transaction request, the transaction should be started straight away. The rules are
$r2 : holds(query-requested(x), res(query(x),s)) \leftarrow$,
$r3 : holds(trans-start(x),s) \leftarrow holds(query-requested(x),s)$,
For a update transaction, after MH has a write request, the lock need to be acquired firstly to start this transaction. The transaction will start after the lock is available.
$r4 : holds(update-requested(x), res(write(x),s)) \leftarrow$,
$r5 : holds(trans-start(x),s) \leftarrow holds(update-requested(x),s)$,
$holds(locked(x), res(acquire-lock(x), res(write(x),s)))$,

Sleep: For a voluntary sleep, the MH informs the proxy of its intention to sleep, and then flushes its dirty pages, gives up any write-locks it holds, after these, the MH goes to voluntary sleep.
$r6 : holds(vol-slept(x), res(release-lock(x), res(flush(x),s))) \leftarrow holds(sleep-sig(x),s)$,
For an involuntary sleep, we suppose the MH is holding a write-lock when it goes to involuntary sleep. In the meantime, if the lock is asked by another writer, the HS will forward the request to the proxy, and the proxy will forward it to the MH. If proxy does not receive the lock from the MH in a limited amount of time, it invalidates the lock and sends it back to HS. And therefore, the MH will inevitably lose the updates it had made.
$r7 : holds(invol-slept(x),s) \leftarrow$,
$r8 : holds(lock-cancelled(x),s) \leftarrow holds(invol-slept(x),s), holds(timeout(x),s)$,
$r9 : holds(update-lost(x),s) \leftarrow holds(lock-cancelled(x),s)$,

Wake up: Upon the MH waking up, the MH sends wakeup notification to MSS and requests missed messages.

$r10 : holds(message-received(x), res(fetch-message(x),$
$res(request-message(x), s))) \leftarrow holds(wakeup-sig(x), s),$
Move/handoff: When the MH notices that it is in a different region, it contacts the new MSS. After new MSS contact HS and old MSS, the MSS proxy broadcasts any invalidations whose timestamp is later than last-time-MSS-contacted-MH.
$r11 : holds(registered(y2, x), s) \leftarrow holds(move\text{-}sig(y2, x), res(move(y2, x), s)),$
$r12 : holds(message-received(y2, x), s) \leftarrow holds(registered(y2, x), s),$
Transaction Commit: After MH requests a transaction, the transaction will be committed or aborted on the HS according to the two phase commit protocol. After that HS sends transaction result to MSS, the MSS broadcasts the transaction result to MH, the MH updates the local knowledge base accordingly based on transaction commit or abort. If after a period time (timeout1) of transaction starting, the MH host still hasn't got any transaction commit or abort notice from MSS for whatever reason, we use closed world assumption in this case, assume not p as $\neg p$ at this time, unkown (not) become no ($\neg$) for the transaction.
$r13 : holds(knowledge-updated(x), res(update-knowledge(x), s)) \leftarrow$
$holds(commit-broadcasting(y, x), s), holds(trans-start(x), s),$
$r14 : \neg holds(knowledge-updated(x), s) \leftarrow holds(abort-broadcasting(y, x), s),$
$holds(trans-start(x), s),$
$r15 : \neg holds(knowledge-updated(x), s) \leftarrow$
$notholds(commit\text{-}broadcasting(y, x), s), notholds(abort\text{-}broadcasting(y, x), s),$
$holds(timeout1(x), s), holds(trans-start(x), s).$

Mobile Support Station (MSS) level:
Register: After the MH registers with MSS, MSS creates the proxy process which retrieves the MH profile from HS, the proxy receives and caches the MH profile, and then broadcasts the sub-profile to the MH, and marks the in-MH-cache bit for those pages. The rules for these are
$r1 : holds(proxy(y, x), res(create\text{-}proxy(y, x), s)) \leftarrow holds(registered(y, x), s),$
$r2 : holds(cached(y), res(cache(y), res(retrieve(y), s))) \leftarrow holds(proxy(y, x), s)$
$r3 : holds(broadcasting(y, x), res(broadcast(y, x), res(retrieve(y), s))) \leftarrow$
$holds(proxy(y, x), s),$
$r4 : holds(marked(y), res(mark(y), s)) \leftarrow holds(broadcasting(y, x), s),$
Start a query or update transaction: After MH requests a query or update, MSS submits this transaction request to HS on behalf of MH. If it is a write request, the lock needs to be acquired firstly to submit this transaction.
$r5 : holds(trans\text{-}submitted(y), res(submit(y), s)) \leftarrow holds(query\text{-}requested(x), s),$
$r6 : holds(trans-submitted(y), res(submit(y), s)) \leftarrow holds(locked(x), s),$
$holds(update-required(x), s)),$
Sleep: For the voluntary sleep, the proxy updates MH-sleep-time and buffers messages and invalidations for the MH until the MH wakes up and is ready to receive them.
$r7 : holds(buffered(y), res(buffer(y), res(update-sleeptime(y), s))) \leftarrow$
$holds(vol-slept(y, x), s).$
In the involuntary sleep case, the proxy doesn't know that MH is not listen-

ing and continues to broadcast invalidations as normal. If the MH is holding a write-lock when it goes to involuntary sleep, in the meantime the lock is asked by another writer, the proxy forwards this request to the MH. If proxy does not receive the lock from the MH in a limited amount of time, it invalidates the lock and sends it back to HS.

$r8 : holds(page - broadcasting(y,x), res(page - broadcast(y,x), s)) \leftarrow$,

$r9 : holds(lock - cancelled(y,x), res(cancel - lock(y,x), s)) \leftarrow$
$holds(invol - slept(x), s), holds(timeout(x), s))$.

Wake up: Upon the MH waking up, the MH sends wakeup notification to MSS and requests missed messages, the MSS then boradcasts missed messages to MH.

$r10 : holds(page - broadcasting(y,x), res(page - broadcast(y,x), s)) \leftarrow$
$holds(wakeup - sig(x), s)$.

Move/handoff: After MH contacts and registers in the new MSS, the new MSS contacts the old MSS to get the MH proxy status. The old MSS flushes any dirty pages to HS and sends the proxy status to the new MSS. The new MSS proxy contacts HS to tell it where to contact the MH. The new MSS proxy broadcasts any invalidations to MH.

$r11 : holds(proxy(y2,x), res(create - proxy(y2,x), s)) \leftarrow$
$holds(registered(y2,x), s)$,

$r12 : holds(flushed(y1), res(flush(y1), s)) \leftarrow holds(move - sig(y2,x), s)$,

$r13 : holds(page - broadcasting(y2,x), res(page - broadcast(y2,x), s)) \leftarrow$
$holds(proxy(y2,x), s)$,

Transaction Commit: After MSS gets the transaction commit or abort notice from HS, the MSS will broadcast the transaction result to MH accordingly.

$r14 : holds(commit - broadcasting(y,x), res(broadcast(y,x), s)) \leftarrow$
$holds(commit - noticed(z,y), s)$,

$r15 : holds(abort - broadcasting(y,x), res(broadcast(y,x), s)) \leftarrow$
$holds(abort - noticed(z,y), s)$.

Home Server (HS) level:

Register: At registration stage, HS sends the pages in the MH profile, marks the MH as a valid reader of those pages, and notes where to contact the MH.

$r1 : holds(sent(z,y), res(send - page(z,y), s)) \leftarrow holds(proxy(y,x), s)$,

$r2 : holds(MH - marked(z,y), res(mark - MH(z,y), s)) \leftarrow holds(sent(z,y), s)$,

Start transaction: After MH requests a query or update transaction and MSS submits this transaction request to HS, the HS starts the transaction. In update transaction situation, MSS submits transaction only when lock is available.

$r3 : holds(trans - start(z), res(do - trans(z), s)) \leftarrow holds(trans - submitted(y), s)$.

Sleep: The sleeping MH process may not return (e.g., MH dies, leaves cell), in this case the proxy may wait around aimlessly. To remedy this problem, the MH status is sent to HS after a system-specific amount of time and the proxy process is killed.

$r4 : holds(proxy - killed(z,y), res(kill - proxy(z,y), s)) \leftarrow holds(timeout(x), s)$,
$holds(vol - slept(x), s)$,

$r5 : holds(proxy - killed(z,y), res(kill - proxy(z,y), s)) \leftarrow holds(timeout(x), s)$,
$holds(invol - slept(x), s)$.

Move/handoff: After MH moves to new MSS, the old MSS will flush any dirty pages to HS and new MSS will contact HS to get these dirty pages regarding the MH.
$r6 : holds(sent(z, y2), res(send - dirtypage(z, y2), s)) \leftarrow holds(proxy(y2, x), s).$
Transaction Commit: According to the two phase commit protocol, if all involved MHs agree to commit the transaction, then the transaction will be committed. If any of them does not agree with commit and want to abort the transaction, then transaction will be aborted. If after a period time (timeout2) the transaction is still not be agreed to be committed, then we use closed world assumption here to assume the transaction won't be committed any more, unknown (*not*) becomes no ($\neg$) in this scenario. For example in the case one of the involved MH has gone to voluntary or involuntary sleep and therefore no commit agreement can be available from that MH during this time duration. After transaction has been committed or aborted, the HS will send transaction commit or abort notice to MSS.
$r7 : holds(trans - committed(z), res(commit(z), s)) \leftarrow$
$hold(commit - agreed(z), s), notholds(abort - agreed(z), s),$
$holds(trans - start(z), s),$
$r8 : \neg holds(trans - committed(z), res(commit(z), s)) \leftarrow$
$holds(abort - agreed(z), s), holds(trans - start(z), s),$
$r9 : \neg holds(trans - committed(z), s) \leftarrow notholds(trans - committed(z), s),$
$holds(timeout2(z), s), holds(trans - start(z), s),$
$r10 : holds(commit{-}noticed(z, y), res(notice{-}commit(z, y), s)) \leftarrow hold(trans{-}committed(z), s),$
$r11 : holds(abort - noticed(z, y), res(notice - abort(z, y), s)) \leftarrow \neg hold(trans - committed(z), s).$

5 A Transaction Example

In this section, we will give an example to demonstrate how to use our logic programming based transaction processing language and model to study transactions in mobile environments. We raise an example to cover the following three different scenarios:
Scenario 1: The MH requests an update transaction, the transaction is committed on HS using two phase commit protocol. The HS sends commit notice to MSS, MSS then broadcasts the commit result to MH, the MH updates the local knowledge base accordingly. This is yes scenario for an update transaction.
Scenario 2: The MH requests an update transaction, the transaction is aborted on HS using two phase commit protocol. The HS sends abort notice to MSS, MSS then broadcasts the abort result to MH, the MH knows that no update should be done in local knowledge base in this scenario. This is no scenario for an update transaction.
Scenario 3: The MH requests an update transaction, but MH hasn't received any commit or abort notice within certain time. This is unknown scenario for an update transaction.

Now we go through our example to discuss these three scenarios. We use our proposed logical programming based knowledge Transaction Processing Language (TPL) and model in section 4 as our restriction language and model here by replacing x, y, z with MH, MSS, and HS respectively. We assume there is a local knowledge base on MH. Let S_0 be the initial state, and suppose we are given that
$t1 : holds(registered(MSS1, MH), S_0) \leftarrow$.
The MH has registered in MSS1. Scenario 1: The MH requests an update transaction. According to MH level $r4$ and $r5$ in section 4, we have
$t2 : holds(update - requested(MH), res(write(MH), S_0)) \leftarrow$,
$t3 : holds(trans{-}start(MH), S_0) \leftarrow holds(locked(MH), res(acquire{-}lock(MH), res(write(MH), S_0))), holds(update - requested(x), S_0))$ After MH requests an update transaction and the lock has been acquired, the transaction will start. MSS1 will submit this transaction request to HS. According to MSS level $r6$ in section 4, we have
$t4 : holds(trans - submitted(MSS1), res(submit(MSS1), s0)) \leftarrow$
$holds(locked(MH), S_0), holds(update - required(MH), s0))$.
As long as MSS submits the transaction to HS, the HS will start the transaction. According to HS level $r3$ in section 4, we have
$t5 : holds(trans - start(HS), res(do - trans(HS), s0)) \leftarrow$
$holds(trans - submitted(MSS1), S_0)$. Then the transaction is committed according to the two phase commit protocol. The HS sends commit notice to MSS. According to the HS level $r7$ and $r10$, we have
$t6 : holds(trans - committed(HS), res(commit(HS), s0)) \leftarrow hold(commit - agreed(HS), S_0), notholds(abort{-}agreed(HS), S_0), holds(trans{-}start(HS), S_0)$,
$t7 : holds(commit{-}noticed(HS, MSS1), res(notice{-}commit(HS, MSS1), s0)) \leftarrow hold(trans - committed(HS), S_0)$.
The MSS broadcasts the commit result to MH. According to MSS level $r14$ in section 4, we have
$t8 : holds(commit{-}broadcasting(MSS1, MH), res(broadcast(MSS1, MH), S_0)) \leftarrow holds(commit - noticed(HS, MSS1), S_0)$.
The MH will update the local knowledge base accordingly after the MSS broadcasts the commit result to MH. According to the MH level $r13$ in section 4, we have $t9 : holds(knowledge{-}updated(MH), res(update{-}knowledge(MH), S_0)) \leftarrow holds(commit - broadcasting(MSS1, MH), S_0), holds(trans - start(MH), S_0)$.
Scenario 2: In scenario 2, after MH requests an update transaction, the transaction will follow the same rule $t2, t3, t4$, and $t5$ as scenario 1. Then the transaction is aborted according to the two-phase commit protocol. The HS sends abort notice to MSS. According to the HS level $r8$ and $r11$, we have $t6 : \neg holds(trans - committed(HS), res(commit(HS), S_0)) \leftarrow$
$holds(abort - agreed(HS), S_0), holds(trans - start(HS), S_0)$,
$t7 : holds(abort - noticed(HS, MSS1), res(notice - abort(HS, MSS1), S_0)) \leftarrow \neg hold(trans - committed(HS), S_0)$.
The MSS broadcasts the abort result to MH. According to MSS level $r15$ in section 4, we have

$t8 : holds(abort - broadcasting(MSS1, MH), res(broadcast(MSS1, MH), S_0))$
$\leftarrow holds(abort - noticed(HS, MSS1), S_0)$.
The MH will know the local knowledge base shouldn't be updated after the MSS broadcasts the abort result to MH. According to the MH level $r14$ in section 4, we have
$t9 : \neg holds(knowledge - updated(MH), S_0) \leftarrow$
$holds(abort - broadcasting(MSS1, MH), S_0), holds(trans - start(MH), S_0)$.
Scenario 3: In scenario 3, after MH requests an update transaction, the transaction will follow the same rule t2, t3, t4, and t5 as scenario1. But after a certain period time (time1), the MH still hasn't got any transaction commit or abort notice from MSS. we use closed world assumption in this case, assume not p as $\neg p$ at this time, unkown (not) become no ($\neg$) for the transaction. According to the MH level $r15$ in section 4, we have
$t6 : \neg holds(knowledge - updated(MH), S_0) \leftarrow$
$notholds(commit - broadcasting(MSS1, MH), S_0)$,
$notholds(abort - broadcasting(MSS1, MH), S_0), holds(timeout1(MH), S_0)$,
$holds(trans - start(MH), S_0)$.

6 Conclusion

In this paper, we proposed a new environment model that integrates the features of mobile environments and intelligent agents. We formalized a logic programming based Transaction Processing Language (TPL) and model to study knowledge transaction processing in mobile environments. We illustrate a typical transaction example to demonstrate how to apply our transaction processing language and model in practical transaction scenarios. In the future, we will use this knowledge transaction language and model to study distributed knowledge transactions and knowledge base in mobile environments.

References

1. Baral, C. and Gelfond, M., Logic Programming and Knowledge Representation. *Journal of Logic Programming* (1994) 73-148.
2. Barbara, D. and Imielinski, T., Sleepers and Workaholics: Caching Strategies in Mobile Environments. In *Proceedings of ACM-SIGMOD 1993 International Conference on Management of Data*, pp1-13, 1994.
3. Blaybrook, B., *On Line Transaction Processing Systems*. John Wiley & Sons, 1992.
4. Chan, W.K. and Chen J., Serializability and Epsilon Serializability in a Mobile Environment. In *Proceedings of seventeenth IASTED International Conference in Applied Informatics*, pp 273-297, 1999.
5. Gelfond, M. and Lifschitz, V., Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* (1991) 365-385.
6. Gray, J. and Reuter, A., *Transaction Processing: Concepts and Techniques*. Morgan Kaufman, 1993.
7. Imielinski, T. and Korth, H.F., *Mobile Computing*, Kluwer Academic Publishers, 1996.
8. Milojicic, D., Mobile Agent Applications. In *IEEE Concurrency* (1999) 80-90.

9. Mirghafori, N. and Fontaine, A., A Design for File Access in a Mobile Environment. In *Proceedings of the IEEE - Conference on Mobile Computing*, pp 57-61, 1995.
10. Weiss G., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.