PoliPer: Policies for Mobile and Pervasive Environments

Luís Veiga and Paulo Ferreira
INESC-ID/IST, Distributed Systems Group, Rua Alves Redol N. 9,1000-029 Lisboa, Portugal
paulo.ferreira||luis.veiga@inesc-id.pt

ABSTRACT

The need for sharing is well known in a large number of distributed applications. These applications are difficult to develop either for fully wired or mobile wireless networks. Such difficulty arises not only because of slow and unreliable connections in such networks but also due to the great diversity of usage scenarios (even for a single application). Currently, programmers are forced to deal with system-level issues such as replication, consistency, security, etc.

PoliPer is a middleware platform, capable of providing the needed flexibility for application development and runtime adaptability. This way, applications can cope with the multiple requirements and usage diversity found in mobile settings. PoliPer relies strongly on the following features: i) the extensible capability to support the specification and enforcement of runtime management policies; ii) a plug-able set of basic mechanisms supporting object replication, security, distributed transactions and code relocation; iii) a set of pre-defined policies that control the mechanisms previously mentioned.

Previous systems are either less comprehensive (they address fewer - or just one - aspects than PoliPer), or less flexible (they are not adaptive). A preliminary prototype of PoliPer has been implemented and evaluated with encouraging results.

Keywords

C.2.4 [Distributed Systems]: Distributed Applications; D.2.12 [Interoperability]: Distributed Objects

General Terms

Design, Languages

1. INTRODUCTION

Due to their intrinsic nature, execution environments, in mobile and pervasive computing, suffer from great and diverse variations during application execution. These variations can either be of qualitative (e.g., network connection or disconnection, specific devices like printers in device neighborhood, consistency and security constrains) or quantitative nature (e.g., amount of usable bandwidth, memory, power available). Applications should be able to deal with this variability of execution environments. However, application programmers should not be forced to account for every possible scenario in their coding. This is unfeasible for two main reasons: i) it is error-prone and difficult to cover all potential situations, and ii) even if correctly performed, it is highly inefficient w.r.t. productivity. Furthermore, these changes often deal with issues that deviate programmers from what they are supposed to do: application-logic. Programmers should not explicitly code non-functional concerns or aspects.

Therefore, this goal can only be achieved through automatic adaptation of applications and adaptation of the execution environment itself. Reflective and adaptive middleware aims at solving these issues by: i) mediating changes in the environment in a manner easily handled by applications, and ii) reacting to changes by reconfiguring itself (either its code, status, module organization) in order to effectively respond to changes.

To address these issues, we propose an architecture fully based on the definition, enforcement and application of declarative XML-defined policies. The contribution of this paper is PoliPer, an extensible policy-driven architecture for mobile and pervasive environments unifying management of several runtime aspects, components and services. These include: transactional support, replication, application deployment, security, and resource management.

The rest of this paper is organized as follows. The next Section presents the global architecture of PoliPer. Section 3 describes policy usage in PoliPer. Section 4 presents a prototype implementation, test scenario and performance results. In Section 5 we compare our work with others and Section 6 draws some conclusions.

2. ARCHITECTURE

PoliPer is a middleware platform capable of providing the needed flexibility for application development and runtime adaptability, so that applications can cope with the multiple requirements and usage diversity found in mobile settings. To achieve this, PoliPer supports the specification and enforcement of a wide set of policies that control a set of basic mechanisms. With PoliPer an application programmer only has to worry with the so-called "business-logic". The adaptability of applications to the particular running scenario (resources available, consistency or security constraints, etc.) is ensured automatically by PoliPer based on the policies provided.

PoliPer relies strongly on the following features: i) the extensible capability to support the specification and enforcement of runtime management policies; ii) a plug-able set of basic mechanisms supporting object replication, security, distributed transactions and code relocation; iii) a set of pre-defined policies that control the mechanisms previously mentioned.

PoliPer architecture is presented in Figure 1. Policies are stored

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
3rd Workshop on Adaptive and Reflective Middleware Toronto, Canada
Copyright 2004 ACM 1-58113-949-7 ...$5.00.

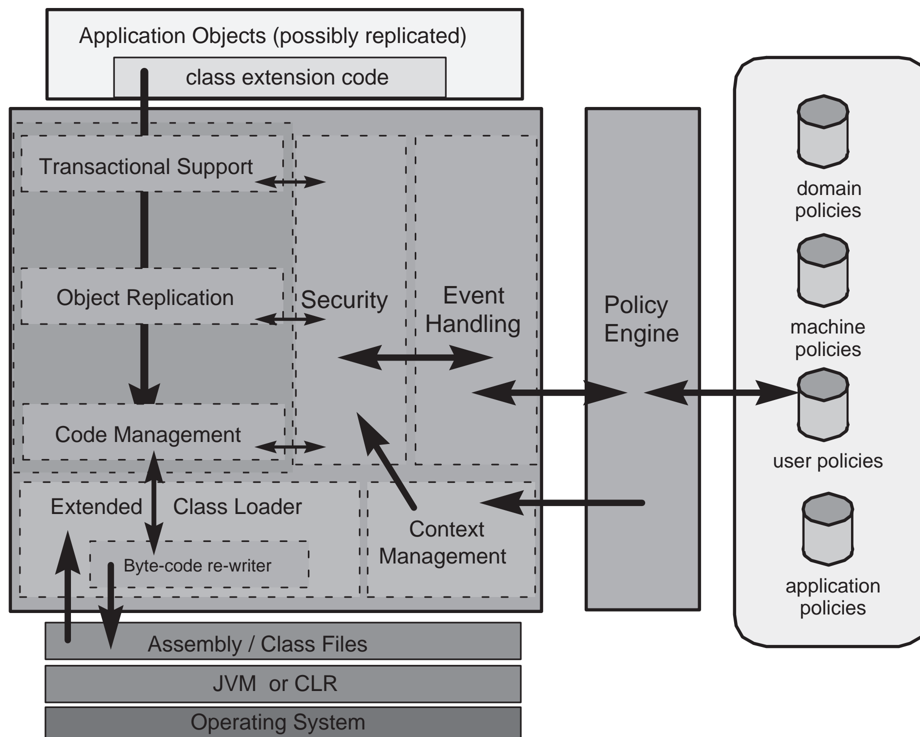

Figure 1: **PoliPer architecture. Transactions, replication and code relocation are policy-driven. Policies are enforced and mediated through event-handling. Security module validates every action performed.**

and categorized by nature. A policy engine receives events generated by PoliPer modules and applications, evaluates policy rules and triggers events, handled by actions based on evaluation results. Every action is validated by a security module, itself, also policy-driven. Code relocation uses an extended class-loader integrated with byte-code modification capability. Object replication and transaction management are performed according to specified policies.

2.1 Policy Management

The policy engine is the main inference component that triggers or mediates responses to events occurred in the system. Apart from all the other existing (and possibly new) modules, security performs a special role, since it must be enforced by auditing or inspecting every system interaction.

The policy engine holds a variable set of policies to be enforced in the system. Policies manage, in abstract, entities. Entities are organized in an open, extensible, namespace-based hierarchy. The entity set includes resources, properties, events, user data and context information. Examples of entities are:
resource.network.connectivity.bluetooth,
property.transaction.optimistic,
event.replication.replicate-in.object.begin.

This hierarchy allows easy management of related entity-sets (e.g., *resource.network.**). Furthermore, entity-groups can be referred to with resort to regular expressions like
event.{*replication,transaction*}*.*.begin*.

A policy is a tuple: {**R**ules, **P**roperties, **E**vents, **A**ctions}.

Rules manage property changes, event triggering and handling with appropriate actions. The definition of a rule must include:

- a domain: a set of entities it relates to.
- a condition of applicability: a custom-predicate to further filter rule application.
- an event to be triggered when the rule domain and condition of applicability are met.

By decoupling the domain, the condition of applicability and the actual action-code, the system can adapt to changes in the environment and modify its own response accordingly. The difference between domain and condition of applicability stems from static versus dynamic analysis that is performed in each case.

Properties are entities with associated value (variable or not). Events are specified by the policy and registered in the event-handling module. Actions can be methods or code snippets, normally predefined event-handlers.

2.2 Event Handling

In PoliPer, notifications to applications and to the various system modules, are performed with resort to events. Provided the necessary permissions, events can be defined either by policies (mainly) or by applications. Events can also be triggered either by applications, by the system modules, or by policies when rules are evaluated. Actions performed, when events are triggered, allow PoliPer and applications, to adapt to changes in the execution environment.

An event is a triple: {**N**ame, **S**ource, **U**serData}.

As entities, events are organized in namespaces. As an advantage, it allows event-handlers defined in policies to listen to specific events, as well as a whole family level of related events, e.g.:
event.replication.replicate-in.object.begin,
event.replication.replicate-in.object.end or
*event.replication.replicate-in.**

Thus, events are organized in a meaningful, yet open manner. It enables regular expression definition of events to subscribe to. This way, event names need not be fully known, or indicated exhaustively by the listener. Nevertheless they are intercepted and handled by the subscribers.

Event jitter can be regarded as bumpiness in the continuous triggering of events, possibly with contradictory response actions. This phenomenon is frequent in execution environments (such as with mobile and pervasive ones) with frequent changes in resources and QoS available to applications. These changes can trigger possibly contradictory measures and with short periods of time between them. To react to them too soon, too often, may hinder system performance and application behavior. In PoliPer, this may be avoided with resort to properties (system, user or application defined) that are evaluated both in the condition of applicability of the rule itself, and possibly updated in the action handling the corresponding triggered event. This process effectively filters events to the degree of stability desired by applications, simplifying application-logic. A straightforward example is a situation of intermittent connectivity where an application is constantly being notified that connectivity is on, and then off. If the application needs a period of stable connectivity, it can use a policy that monitors connectivity-related properties. Policy actions hide these quick variations, and notify the application only when there is minimum signal strength or, in alternative, when some delta time has elapsed since the last time connectivity was on.

2.3 Context Management

In PoliPer, there is a context management module. This module performs resource abstraction and manages properties whose values vary during execution. Abstraction enables representing physical machine resources as sets of primitive context properties. Examples include memory, connectivity, bandwidth available, etc. For flexibility, resources, as entities, are also namespace-organized.

The actual mappings between basic/primitive resources and resource designations is performed by the context manager. Each of these resources implies an architecture-dependent way of measuring. This heterogeneity is masked, to the rest of the system, by a low-level component in the context manager.

Situations like appearing devices, discovering remote resources or application counterparts are also handled by the context manager. The relevant properties are updated and the appropriate events are triggered. In more general terms, any change to the properties (resources, middleware state or user-defined properties) managed by the context manager can potentially trigger associated events defined by the policies loaded.

In our work we do not specifically address adapting resources (and possibly replacing them with variants) but solely on representing them, in a flexible manner, and monitor their changing properties. Events to be triggered and actions to address them are described in policies. Appropriate policies configure context management and its events, in order to allow applications to be notified solely when these changes are stable or reach a certain threshold.

PoliPer does not try to manage resources centrally. This is performed by the combination of security policies that monitor resource requests and context management that registers and notifies resource shortage.

3. POLICY-DRIVEN MODULES

On top of the basic architecture, PoliPer specifies a set of predefined policies and police-driven modules that manage specific execution mechanisms.

3.1 Code Management

PoliPer supports the specification and enforcement of policies concerning in which computer an application should run. In particular, based on the hardware and software characteristics available in the neighborhood computers, it allows to decide where an application should run or, even while it is running, to move it entirely (or part of it) to another computer with more resources. These policies may refer to devices explicitly, by categories (e.g. PDA, Laptop, etc.) or, more generally, by demanding that specific capabilities and resources be present.

PoliPer makes extensive use of code creation and modification. This fundamental feature allows both to modify existing instructions and to create new ones; this can be done either when applications are compiled or at runtime. Consequently, on one hand, we can run legacy code on PoliPer (in addition to new applications, obviously) thus taking advantage of all its basic mechanisms (replication, security, etc.) even if such applications were programmed without such in mind. On the other hand, we can modify an application at runtime increasing its adaptability to a particular scenario. In resource-constrained devices like PDA's, this last option is not readily available. To deal with this limitation, the programmer must give hints, at development or compile-time, using declarative attributes, so that special hooks may be inserted in the compiled code. These hooks will then invoke the middleware that will respond according to loaded policies and running context in the device.

3.2 Object Replication

PoliPer supports the specification and enforcement of policies concerning the replication of objects. In PoliPer, object replication is incremental and adaptive. Unless otherwise specified, it is performed transparently to applications but can also be flexibly configured by them. In particular, it allows the specification of:

- the best moment to create a replica.
- when to merge two or more replicas of the same object.
- the amount of objects to replicate at a given time (a cluster in PoliPer).
- which branch of a graph should be further replicated.
- which objects should be swapped-out (i.e. dynamically replaced by a proxy and transfer the remaining objects to a neighboring device).

This last functionality, when performed, may also trigger code management events. This module functionality provides the necessary byte-codes to the device temporary hosting the swapped objects.

The most relevant events are triggered with the replication (either in or out) of each single object:
event.replication.replicate-in.object.begin
event.replication.replicate-in.object.end, and
event.replication.replicate-out.object.begin
event.replication.replicate-out.object.end

Additionally, there are two more sets of events with coarser granularity:
event.replication..cluster.**
event.replication..graph.**

The first set handles clusters (groups of object replicated in a single time as a unit) and the second one addresses complete graph branches. Different granularity of events, triggered at different times, provide a basis for flexible management of different scenarios w.r.t latency/bandwidth tradeoffs and consistency management.

3.3 Transactional Support

PoliPer also supports the definition and enforcement of transaction policies. They provide a flexible and adaptive model for transactional support. This model is more suited to mobile and pervasive environments than the classic ACID model. It allows the specification of transaction behavior w.r.t.:

- alternative sources to fetch objects.
- data consistency degree: whether specific objects, clusters, or graphs, should be fetched from their home nodes, caching nodes, or not required at all for transaction to proceed.
- whether to cache changes made to the data.
- atomicity required to complete transactions: transaction policies can allow, require or discard, at commit time, changes made to specific objects, or to objects fetched from specific locations.
- how failures are handled, e.g., automatically lower some of the transaction demands in order to be able to commit some of the work performed.

Transactional mechanisms rely on a lower-level, and coordinate, replication mechanisms to provide objects to transactions. Examples of specific transaction support events are:
event.transaction.fetch.begin, *event.transaction.commit.fail*.

3.4 Security

PoliPer supports the specification and enforcement of security policies. Security policies are used primarily for authorization purposes and to monitor and limit resource utilization. This includes hardware resources like processing power, network bandwidth, and memory, as well as other software resources like GUI windows and widgets.

As portrayed in PoliPer architecture, the security module also monitors all interactions between event-handling (trusted and linked to the policy engine) and every other module in the system. This allows policies to enforce security in the execution of every aspect of the other mechanisms provided by PoliPer. For instance, they prevent policy deployment and event subscription without the required permissions.

PoliPer also allows the specification of history-based security policies [13] well adapted to mobile agents needs (e.g. preventing abusive resource consumption, enforcing chinese-wall or obligation policies, etc.).

4. IMPLEMENTATION

A prototype implementation of PoliPer has been developed. It runs on .Net (for desktop and laptop nodes) and .Net Compact Framework (for SmartPhone and PocketPC). The primary programming language used is C#. Policies are coded in XML. Since Remoting services are not available in .Net CF, web-services and SOAP is used instead. Desktop machines have Internet Information Services installed, and a small footprint mobile web-server is used in PocketPC. In PoliPer prototype, we extended and integrated some of our previous work regarding: i) security policies [4], ii) transaction policies [14], and iii) adaptive replication on mobile devices [15]. Tests were performed with the following infrastructure: a Pentium 4, 2.8 Ghz, 512 MB PC, an IPAQ 3360 Pocket PC connected through a USB Bluetooth adapter at 700Kbps.

4.1 Evaluation

The prototype qualitative evaluation was done testing application correct functionality w.r.t: security, code deployment, mobility, replication, and optimistic transactional support; all in the presence of various environment changes (connectivity, bandwidth, memory, neighboring devices) triggering policy-defined behaviors.

A vast number of scenarios can be imagined to portray adaptable behavior of applications designed with adaptability in mind. The developer is in charge of determining the situations the application should respond and adapt to. Then, a set of policies should be defined. They will configure the middleware to detect those situations, evaluate conditions of applicability, trigger the appropriate events and run the corresponding actions.

Other relevant scenarios are those of dynamic adaptation, to some extent, of applications that were designed without adaptability in mind. With PoliPer, we can easily set up, for this purpose, the following example scenario.

A number of applications are running. They have already replicated some data for local disconnected use. Application code simply navigates through object graphs; it is not aware of PoliPer (remember the applications were not designed with adaptability in mind). The following policies are loaded and acting/reacting as follows, with increasing priority:

- Policy P_1 determines, for each application and according to the available bandwidth, the size/depth of each replication cluster.
- Policy P_2 determines that whenever connectivity is back on, and the application has accessed a threshold fraction of the previously replicated objects, another cluster of objects should be immediately pre-fetched.
- Policy P_3 determines that when there is Bluetooth connectivity, GPRS access should not be used, for economic reasons.
- Policy P_4 determines that whenever a threshold value of communication cost has been reached, the user should be advised and pre-fetching should be disabled. From then on, objects should only be replicated on-demand by applications.

This set of policies could be installed in the system: i) by default, ii) declaratively defined by an application programmer, iii) setup by a system administrator, or iv) created by the user through an interactive policy generation tool. This example clearly shows a situation where policies can dynamically manage middleware and application execution, without the need to write new application code for adaptability to each scenario. In the example, application code was simply extended in order to allow incremental replication. More sophisticated behavior simply emerges from the concurrent enforcement of this set of policies by PoliPer.

4.2 Performance

System performance was measured with the following micro-benchmark addressing a specific aspect: object replication. In the test, after an object replication policy was loaded in the system, series of iterations were executed on a list of hypothetical appointments with 300 elements with different payloads: 64 and 1024 bytes each. Therefore, as the list of appointments is iterated, in each element object, an empty method is invoked.

When an object is not yet replicated, the respective events are triggered by its proxy, and the replication mechanism takes over. It replicates the object where the fault occurred. Additionally, a policy-configurable number of other objects is also pre-fetched. Such a set of objects is called a cluster in PoliPer. In the end of

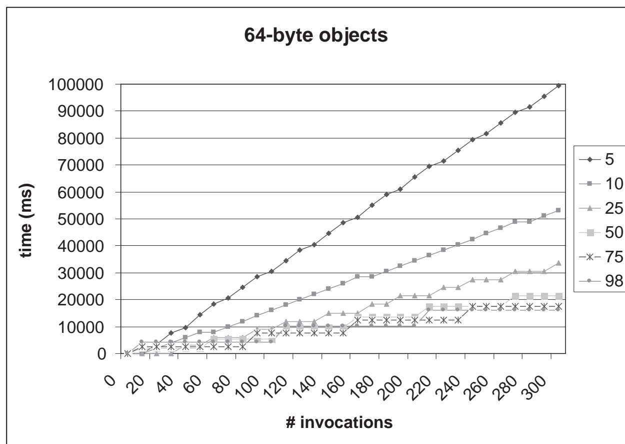

Figure 2: Performance results of incremental replication of objects with 64-byte payload.

each test, 300 objects have been replicated. The replication mechanism was configured, by means of different policies, to replicate objects, on-demand, with a depth of 5, 10, 25, 50, 75 and 98 objects each time.

The graph in Figure 2 show that replication performance is mostly latency-bound. It is more efficient when several (more than 25) objects are replicated each time. Overhead due to policy enforcing is negligible and dominated by communication. Therefore, the increased flexibility imposes very little cost w.r.t performance. Additional results show that when object payload is raised from 64 to 1024 bytes (a sixteenfold increase), replication performance drops only 60%.

5. RELATED WORK

Concerning the flexibility of the programming paradigm, PoliPer can be related to several other middleware systems. With respect to mobile transactions, most research considers networks where mobile hosts connect via wireless to fixed base stations [1, 12, 16]. These solutions are typically client-server based and do not address the adaptability needs of applications so that they can cope with the multiple requirements and usage diversity found in mobile settings.

However, there are mobile transaction systems that use semantic information to adapt the behavior of transactions. For example, in Pro-Motion [16], data is encapsulated in *compacts* allowing the definition of consistency rules to be applied to such data set as a whole. In Clustering [11], it is possible to specify consistency degrees among replicated data. Moflex [8] also provides a mechanism for describing the associated behavior while crossing wireless cells. With Toggle [5], it is possible to specify different atomicity and isolation degrees, by dividing a transaction in vital and non-vital sub-transactions.

One of the most important works addressing security is Ponder [6]. It provides a general-purpose deployment model for security and management policies. Its declarative language is able to express and specify some generic and complex security policies such as role based access control (RBAC) policies. In particular, the obligation policies provided by Ponder are used in an agent platform [9] to specify mobility policies of agents. In this platform application logic is completely separated from migration logic. Although designed for mobile agents, this platform still does not consider any type of support for history-based security (e.g. preventing abusive resource consumption, enforcing a chinese-wall policy, etc.) which we believe to be important for mobile applications.

In PoliPer, system entities like policies, properties and events are defined in a namespace-based hierarchy combined with regular expressions. A related approach is used for security policy files in the Java language. In Java, permissions are defined in a class hierarchy. Access to system properties can be managed using wildcard substitution when referring to property names. Java policy files are designed solely for security purposes, i.e., granting or denying access to resources. This approach is rather limitative for the purpose of this work. In PoliPer, the range of policy use is broader. Furthermore, the middleware and applications can react to changes in the system (resource management, access control, etc.) with definable programmatic actions.

In [7], a re-configurable reflective middleware platform is defined to meet varying transactional requirements from applications, by supporting concurrently running transaction services.

QoS non-functional aspects are extracted and defined declaratively by contracts in [2]. Compatible contracts are combined straightforwardly. When conflicts among requirements from different contracts arise, they are solved based on priority. Connectors effectively externalize interactions and associations between objects. In PoliPer, a contract can be regarded as a set of policies.

In [3], a publish/subscribe messaging model is presented, defining channels in a hierarchical manner. This hierarchy is reflective, dynamic and de-coupled from publishers and subscribers. As a consequence, there is no guarantee that a specific channel will exist. PoliPer uses a namespace-based organization for entities (including events), that can be accessed hierarchically and, for increased flexibility, with resort to regular expression matching. In PoliPer, event publishers and subscribers are also fully de-coupled.

In [10], resources are encapsulated and managed by an extensible framework. When necessary, resources are dynamically adapted within the middleware to suit each requiring task. This operation is performed through negotiation. The middleware keeps track of the associations among resources and tasks. Resource and context representation in PoliPer follows a related approach. Nonetheless, in PoliPer, the focus is somewhat different. Instead of resource management and adaptation, it aims at adapting to resource variations.

In summary, previous systems are either less comprehensive (they address fewer - or just one - aspects than PoliPer), or less flexible (they are not adaptive). PoliPer comprehensively addresses important aspects as object replication, transactional management, code deployment and security, with a fully open policy-driven approach.

6. CONCLUSION

The main contribution of this work is PoliPer itself: a middleware platform that helps programmers to develop distributed mobile applications, by allowing them to focus on the application logic.

With PoliPer, system-level issues such as object replication, abusive resource consumption by mobile agents, transactional support, etc. are automatically handled by the system. Most importantly, the behavior of such basic mechanisms is policy-specified and automatically enforced, making applications widely adaptable and flexible to different usage scenarios.

A prototype implementation was developed and its performance evaluated. The preliminary results are encouraging. In the future, we intend to conclude and optimize PoliPer implementation, and to port it to Java-based architectures.

7. REFERENCES

[1] Naser S. Barghouti and Gail E. Kaiser. Concurrency control

in advanced database applications. *ACM Computing Surveys*, 23(3):269–317, 1991.

[2] R. Cerqueira, S. Ansaloni, O. Loques, and A. Sztajnberg. Deploying non-functional aspects by contract. In *The 2nd International Workshop on Reflective and Adaptive Middleware, Middleware 2003*, Rio de Janeiro, Brazil, june 2003.

[3] E. Curry, D. Chambers, and G. Lyons. Introducing reflective techniques to message hierarchies. In *The 2nd International Workshop on Reflective and Adaptive Middleware, Middleware 2003*, Rio de Janeiro, Brazil, june 2003.

[4] Pedro Dias, Carlos Ribeiro, and Paulo Ferreira. Enforcing history-based security policies in mobile agent systems. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, jun 2003.

[5] R. A. Dirckze and L. Gruenwald. A toggle transaction management technique for mobile multidatabases. In *Proceedings of the CIKM 98*, pages 371–377, Bethesda, MD, USA, 1998.

[6] N. Dulay, E. Lupu, M.Sloman, and N. Damianou. A policy deployment model for the Ponder language. In *7th IEEE/IFIP International Symposium on Integrated Network Management*, Seattle, USA, 2001. IEEE press.

[7] Randi Karlsen and Anna-Brith Jakobsen. Transaction service management: An approach towards a reflective transaction service. In *The 2nd International Workshop on Reflective and Adaptive Middleware, Middleware 2003*, Rio de Janeiro, Brazil, june 2003.

[8] Kyong-I Ku and Yoo-Sung Kim. Moflex transaction model for mobile heterogeneous multidatabase systems. In *Proceedings of the 10th International Workshop on Research Issues in Data Engineering*, San Diego, California, 2000.

[9] Rebecca Montanari and Gianluca Tonti. A policy-based infrastructure for the dynamic control of agent mobility. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, Monterrey (USA), June 2002.

[10] N. Parlavantzas, G. Coulson, and G. Blair. A resource adaptation framework for reflective middleware. In *The 2nd International Workshop on Reflective and Adaptive Middleware, Middleware 2003*, Rio de Janeiro, Brazil, june 2003.

[11] Evaggelia Pitoura and Bharat K. Bhargava. Data consistency in intermittently connected distributed systems. *Knowledge and Data Engineering*, 11(6):896–915, 1999.

[12] K. Ramamritham and P. K Chrysanthis. A taxonomy of correctness criterion in database applications. *Journal of Very Large Databases*, 4(1), 1996.

[13] R. Sandhu. Separation of duties in computarized information systems. In *Proceedings of the IFIP WG11.3 Workshop on Database Security*, Halifax, UK, September18–21 1990.

[14] Nunos Santos, Luís Veiga, and Paulo Ferreira. Transaction policies for mobile networks. In *5th IEEE International Workshop on Policies for Dist. Systems and Networks(Policy 2004)*, 2004.

[15] Luís Veiga, Nuno Santos, Ricardo Lebre, and Paulo Ferreira. Loosely-coupled, mobile replication of objects with transactions. In *Workshop on Qos and Dynamic Systems. 10th IEEE International Conference On Parallel and Distributed Systems(ICPADS 2004)*, 2004.

[16] Gary D. Walborn and Panos K. Chrysanthis. Supporting semantics-based transaction processing in mobile database applications. In *Symposium on Reliable Distributed Systems*, pages 31–40, 1995.