Mobile Middleware and Transaction Services

Arne Ketil Eidsvik
Department of Computer Science
University of Tromsø

Randi Karlsen
Department of Computer Science
University of Tromsø

Abstract

Transaction execution in mobile environments needs to be flexible to support typical mobile computing characteristics, like movement, disconnections and limited resources, and also to support the variety of transactional properties that might be required by different applications. This paper proposes a solution for flexible transaction processing in mobile applications, and describes a middleware (MobileTSe) that makes different transaction services available on mobile units.

1 Introduction

Mobile computing is characterized by limited resources in devices, movement, and a growing number of mobile applications requiring transaction execution with various properties. Transactions with strict ACID (Atomicity, Consistency, Integrity, Durability) properties will in a mobile environment lead to frequent deadlocks and aborted transactions, due to disconnections and long lived transactions. Various mobile transaction models like [1, 2, 3] have been proposed.

Existing mobile transaction models solve various problems with mobile transaction execution. Some solve the mobility problem, others solve disconnections and so on. Instead of proposing another mobile transaction model, we propose a solution which utilizes the existing mobile transaction models and offer the applications a menu of transactional properties. We believe that a middleware which is capable of adapting to various transaction models and various transactional requirements can provide the required flexibility to support different mobile transaction properties. We presented an architecture for such a middleware in [4] and named it `MobileTSe`.

`MobileTSe` is a component framework in which various *local* transaction services can be deployed. However, because of the limited resources in mobile devices, we see a need for the devices to be able to access also *external* transaction services. An external transaction service is deployed on a separate device from where the application and the `MobileTSe` are installed. A deployed component in `MobileTSe` provides an interface between `MobileTSe` and the external transaction service. This interface is realized with UPnP (Universal Plug and Play) [5, 6]. In this paper we concentrate on the interface between applications and `MobileTSe`.

In section 2 we motivate our work with two scenarios. Section 3 is about some relevant transaction models and in section 4 is a brief overview of the architecture of `MobileTSe`

This paper was presented at the NIK-2006 conference. For more information, see //www.nik.no/.

and an introduction to the interfaces between MobileTSe, applications and transaction services. We summarize related work in section 5 and conclude in section 6.

2 Scenarios

A widely used example of long lived transactions is travel bookings for instance of flight, hotel and car rental. The booking is three separate subtransactions which may be composed into one long lived transaction. Disconnections and movement may occur during a transaction if the booking is done from a mobile device. Furthermore, different transaction models might be required according to the choices of the user. He might decide that either all or none of the bookings are made, or he might decide that the flight shall be booked regardless of success of the hotel- and car-subtransactions.

Information sharing at an academic conference is another example of transaction execution in mobile environments. Conference participants and organizers meet and share information, traditionally through conference presentations and informal chats at the conference area. A mobile information system will facilitate further possibilities for information sharing, and the use of transaction services will enable application programmers and users to secure a relevant level of correctness. Participants can use their PDAs and laptops to search for information from conference organizers and from fellow participants and also offer their own information for sharing (for instance information about themselves, their research projects, employers and a selection of their own articles). Conference organizers will typically provide downloadable conference proceedings, slide presentations, bulletin boards, further information about profiled products from conference exhibitions and offer ticket sales for conference happenings.

In such an information system we will find both long and short transactions, some with ACID properties and some with loosened requirements. An example of a long transaction with loose properties is when a user downloads all available information on a specific topic from both organizers and participants. This transaction could last long enough for the user to move beyond the range of the wireless network and get disconnected. When the user is reconnected the transaction continues transparently. This kind of transaction need not require strict atomicity, but the user should be notified whether the transaction was successful or not.

In other circumstances a conference participant can require transactions requesting information about him and his projects to be atomic. In this way the user can assure that all presented information about him is complete. Transactions involving payments like for instance buying tickets to a conference concert will require complete ACID properties.

Few mobile transaction models have been implemented in commercially available systems. One reason may be the variation in requirements for mobile transactions as illustrated in the above example. Our proposition for a flexible transactional middleware allows for a dynamic use of existing mobile transaction models through adaptation at runtime. Such a middleware will make it more valuable for application programmers to utilize the advantages of fault tolerant transaction services according to application specific requirements and environmental properties.

3 Transaction Models

MobileTSe enables access to transaction services which may range from mobile services, such as transaction services based on the Kangaroo transaction model or the Cluster

model, via models designed specifically for long lived transactions like Saga to services providing strict ACID properties.

The Kangaroo and Cluster transaction models are both influential models that describe mobile transaction executions with different properties. We believe that both models are very useful, and will show how `MobileTSe` can support both of them, together with an extensible number of other transaction services. This section gives a short description of some relevant transaction models.

The Kangaroo model supports movement through Kangaroo transactions which are split into Joey transactions as the unit moves in mobile networks [1]. The model is intended to support mobile units accessing resources in a wired network behind the base stations of a mobile network. Transactions are managed within the base stations of the network. When the mobile unit moves from one base station to another, the Kangaroo transaction is split into Joey transactions, one for each base station. Each Joey transaction is committed and the complete Kangaroo transaction is resolved after the last Joey transaction to commit the complete Kangaroo transaction.

The Saga transaction model [7] was originally designed to manage long lived transactions in fixed networks. However, the Saga transaction model is useful also for mobile environments. A saga is a set of compensateable transactions. Each transaction can be committed and the results made available to other transactions. If one of the transactions is aborted, the rest of the committed transactions in the same saga must be compensated. The Saga transaction model is suitable for the booking scenario if the user wants to either book all or none of the flight, car- and hotel-reservations. If the application is run on a mobile device it might be switched off or leave the network in the middle of a saga. When the mobile device returns back to the network the saga may continue. If he does not, the committed transactions will be compensated. The Saga transaction model has been used in [8] to modify the Kangaroo model to make it more efficient in dealing with long lived transactions over mobile networks.

The Cluster transaction model is intended to support strong transactions within clusters and weak transactions across cluster boundaries [2]. A set of mobile units connected to a reliable and high performance local network can form a cluster. When a unit leaves the network for a period of time it can form its own cluster and continue whatever transaction it is involved in, but now in the form of a weak transaction. When the mobile unit returns back to the network, the transaction is resolved for inclusion in the original, strong transaction.

Each of the mentioned transaction models solve different aspects of mobile transaction execution and as the mobile devices turn more and more into general computers, the applications will also require a wider range of transaction properties.

4 MobileTSe

`MobileTSe` is a flexible transaction service environment. Local transaction service components may be deployed in `MobileTSe` and external transaction services may be controlled from `MobileTSe`. Transaction services may comply with various models to support different transaction requirements.

Architecture

The limited resources in mobile units requires a lightweight implementation of `MobileTSe` while dynamical transactional requirements demand a flexible solution

which might require a larger software systems. This is solved in MobileTSe by allowing a combination of local and external transaction services. Lightweight transaction services may be deployed in MobileTSe on the mobile unit. Other transaction services may be accessed on external units. The former are local transaction services and the latter are external transaction services. The middleware (MobileTSe) in the mobile unit is flexible according to various transaction requirements, interaction types and service discovery protocols. The flexibility is facilitated with service discovery protocols to access external transaction services and a flexible middleware which is able to adapt according to environments and required transaction properties. The demand on hardware resources in the mobile unit is minimized by using external transaction services.

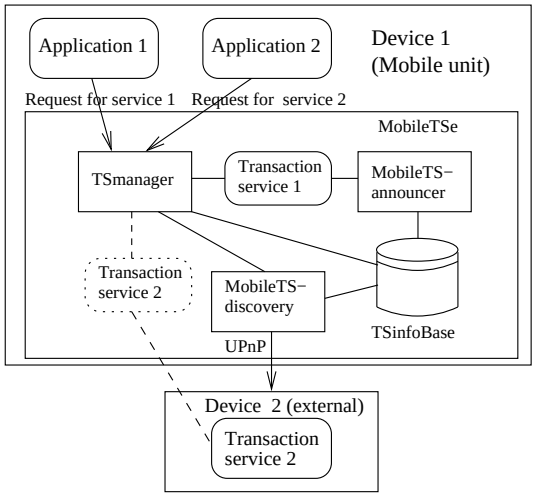

Figure 1: MobileTSe architecture

Figure 1 depicts the architecture of MobileTSe which consists of a TSmanager, MobileTSdiscovery, a MobileTSannouncer and an information base with meta information about available transaction services. MobileTSe can offer a currently deployed local transaction service or an externally available transaction service. When a new transaction service is needed TSmanager first tries to find a suitable service described in the stored meta information. If no service is found, a search is issued through the MobileTSdiscovery and a service discovery interface. The MobileTSdiscovery component presents an external transaction service as a virtual local service to the TSmanager. A virtual local transaction service is shown with dotted lines in figure 1.

When a mobile unit enters a network it will establish which service discovery protocols are used (e.g. UPnP, SLP or Jini), then the interaction type (RPC or publish subscribe) and finally which transaction services are available. The MobileTSdiscovery does a search for available transaction services and stores their properties in the information base. When a request for transaction execution is subsequently received by the MobileTSe, the most appropriate transaction service available is chosen.

The MobileTSannouncer will advertise - and respond to requests for - local transaction services which are made available for other units in the network. In this way MobileTSe can act both as a server and a client in a peer-to-peer network of transaction services.

Our work on flexibility with respect to interaction types and service discovery protocols is based on ReMMoC [9]. ReMMoC is a reflective middleware [10] which can

adapt according to various interaction types and service discovery protocols. MobileTSe has been influenced by TSenvironment [11, 12], which is a component framework where different transaction service components can be deployed, modified and concurrently used. In [13] ReflecTS, a flexible transaction processing platform, and a transaction service selection procedure are presented. In contrary to TSenvironment and ReflecTS, MobileTSe is specifically designed for use in mobile environments.

Interfaces

The interfaces between MobileTSe, applications and transaction services makes MobileTSe act as a layer between applications and transaction services with a single interface towards applications and a tailored interface towards each type of transaction service, for instance services based on ACID, Kangaroo or Saga transaction models. The interface between MobileTSe and applications provides mechanisms for specifying transactional requirements and for interacting with the transaction services. The interface between MobileTSe and transaction services is provided by a set of components, one for each transaction service. A corresponding component is plugged into MobileTSe when a specific transaction service is accessed.

The interface between MobileTSe and transaction services is realized with a service discovery protocol. In our example we have used UPnP because it supports both wireless networks and state eventing making it possible to monitor transaction state. Using UPnP, remote transaction services can be described, detected and controlled. In this paper we focus on the interface between MobileTSe and applications.

A typical transaction service consists of a transaction manager (TM) and a resource manager (RM) for each data source. The X/open specifications of the distributed transactions model (DTP) [14] assumes the software components $application$, RM and TM, and specifies the interfaces between each component: the xa interface between TM and RM, and the tx interface between $application$ and TM. A $service$ interface is used between $application$ and RM.

Existing transaction services provide a variety of interfaces between $application$, RM and TM components. Some are compliant with the X/open tx and xa interfaces, others are not. Details about these interfaces should be hidden from the application, and particularly from applications for mobile environments because these will come across different transaction services as the mobile unit moves. The application needs to be aware of the transactional properties requested as a result of applicational or user requests, not the possible interfaces to all the transaction services which the unit may come across. MobileTSe provides a single interface which comprises the transactional properties required by various applications.

In some cases, for instance when the mobile unit is located within a local network and a short ACID transaction from a stationary transaction service is needed, the interface between MobileTSe and the transaction service will be similar to the solutions in component based middleware for stationary units. If the unit is moving during the transaction execution, however, the middleware should use a model which handles movement, such as the Kangaroo transaction model [1]. In other situations the Cluster model [2] may be needed to support disconnections and weak transactions, or Saga [7] to support long lived compensating transactions.

MobileTSe provides a $Mobile\_TX$ interface to the application. This interface is based on the X/Open DTP specification with extensions for requesting transaction properties. MobileTSe interfaces the various transaction services, hiding details about

the service from the application.

The $MobileTX$ interface between applications and `MobileTSe` contains a basic set of functions as specified by X/Open: tx_begin, tx_close, tx_commit, tx_info, tx_open, $tx\_rollback$, $tx\_set\_commit\_return$ and $tx\_set\_transaction\_timeout$. These are functions present in the interfaces to many transaction managers. The $MobileTX$ interface contains, in addition, functions like req_ACID to request transaction services with ACID properties, req_Saga to request Saga based services and so on. Functions like $req\_disconnectable$, $req\_movable$ and $reg\_compensateable\_sub$ are included to facilitate requests from the application for transaction services with specific properties. $Req\_disconnectable$ is issued to request transaction services which are able to allow for disconnections within a transaction execution, and $req\_movable$ to issue requests for transaction services which support movement. $Rec\_compensateable\_sub$ is issued to request a transaction service which can offer transactions with compensateable subtransactions.

Various extensions to the X/Open DTP specifications of the TX interface are typically supplied by different transaction models. For instance Saga requires mechanisms to: inform the system of the beginning and end of a saga ($begin\_saga$, end_saga), and the beginning and end of each transaction ($begin\_transaction$, $end\_transaction$). It also needs a command to start a user-initiated abort ($abort\_transaction$) to terminate the current transaction and a command to terminate the complete saga ($abort\_saga$). There is also a command to commit the currently executing transaction and to complete the saga (end_saga), and a command ($save\_point$) to let the application programmer indicate check points between transactions in a saga.

Transaction services can be requested *explicitly* with a request like req_Saga or *implicitly*. Implicit requests can either be issued with a request function for certain properties, like for instance $reg\_compensateable\_sub$ or simply by the application starting to use functions only offered by certain types of transaction services, like for instance various Saga commands. Implicit requests may result in different types of transaction services as long as their transactional properties comply with the request.

5 Related Work

Several models for mobile transactions have been proposed, for instance [1, 2, 3], in which disconnects and movement are handled differently. Even though these models allow flexibility with respect to mobile behavior, they do not provide the necessary flexibility with respect to transactional properties. They all describe transaction processing according to a predefined set of properties, and do not have the ability to adapt according to different needs.

Flexible transactional systems are previously proposed in [15, 16]. [15] describe a reflective transaction framework that implements extended transaction models on top of TP-monitors. The framework uses transaction adapters on the meta level to extend TP-monitor behaviour. In [16] Reflective Java is used to implement a flexible transaction service that allows application developers to provide application-specific information to a container so that this can be used to customize the transaction service.

Related research on dynamic combination and configuration of transactional and middleware systems is found in [17, 18, 19, 20]. These works recognize the diversity of systems and their different transactional requirements, and describe approaches to support these needs. However, they do not specifically address mobile systems and do not allow the type of flexibility described in this paper. In [17] a formal method to

synthesize transactional middleware is specified. The work describes an approach that takes transactional requirements for a given system as input, selects available service components and composes a transactional middleware customized to the needs of the system. [18] argue the necessity to allow both design time and runtime specification of transaction models. Transaction model elements are organized so that parts of the specification can be done before transactions are executed, while the remaining parts can be specified during runtime. Runtime specification of transaction executions are done by users. [19] proposes an extension of the transaction concepts in EJB, called Bourgogne transactions, that adds a set of advanced transactional properties allowing some flexibility in transaction executions. In [20] the ACTA framework is used as a tool to support the development and analysis of new extended transaction models. However, implementing a model specified in ACTA is left to the developer.

Web services transaction management, such as [21, 22, 23], is relevant for our work because mobile devices are increasingly used for accessing resources on the Web. Particularly work on composite Web services and transaction management are relevant because different Web services included in a composition may require different transaction properties. The work of [24] describes the WSTx (Web service transactions) framework for building reliable transactional compositions from Web services with diverse transactional behaviour. By using WSDL (Web Service Description Language), Web service providers declare their individual transactional capabilities and semantics, and Web service clients declare transactional requirements. By harmonizing capabilities and requirements reliable transaction executions can be provided.

`MobileTSe` contrasts previous work in the ability to adapt to varying mobile transactional requirements by offering a number of available transaction services, that are both local and external to the mobile unit. Allowing remote transaction execution through transaction service discovery is a novel approach to mobile transaction execution. Our proposition allows adaptation to new transaction models during runtime, and opens up for adaptation to models that did not exist when the application was implemented.

6 Conclusion

We have presented `MobileTSe`, a flexible transaction service framework for mobile environments, which can adapt according to the varying requirements and constraints for mobile transaction execution. `MobileTSe` provides an environment in which a limited number of transaction services can be deployed, and from where a suitable transaction service can be chosen for the execution of a transaction.

`MobileTSe` uses requirements from the application to choose an appropriate transaction service. A qualified search for a suitable transaction service in the network is performed if the chosen set of transaction properties cannot be met by any of the locally available services. If a remote transaction service is found and accepted, transaction management is transparently propagated to the chosen service. The paper describes a novel architecture for remote transaction service access, where tx interface components are used for interfacing remote services.

Transaction services external to the mobile unit are made available in `MobileTSe` through transaction service discovery. In a previous paper [4] we described how UPnP is used for discovery and control of available transaction services in the network.

We have described the interface between applications and `MobileTSe`, and shown how both explicit and implicit requirements from the application are used to choose an appropriate transaction service.

References

[1] Dunham, M.H., Helal, A., Balakrishnan, S.: A mobile transaction model that captures both the data and movement behaviour. Mobile Networks and Applications **2** (1997) 149–162

[2] Pitoura, E., Bhargava, B.K.: Maintaining consistency of data in mobile distributed environments. In: Proceedings of 15th International Conference on Distributed Computing Systems, Vancouver, British Columbia, Canada (1995) 404–413

[3] Walborn, G.D., Chrysanthis, P.K.: Supporting semantics-based transaction processing in mobile database applications. In: Proceedings of the 14th IEEE Symposium on Reliable Distributed Systems. (1995) 31–40

[4] Eidsvik, A.K., Karlsen, R., Blair, G., Grace, P.: Transaction service discovery in mobile environments. In: AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 2 (AINA'06), Washington, DC, USA, IEEE Computer Society (2006) 224–228

[5] UPnP Forum: UPnP device architecture 1.0.1. http://www.upnp.org (2003)

[6] Jeronimo, M., Weast, J.: UPnP Design by Example, A Software Developer's Guide to Universal Plug and Play. Intel Press (2003)

[7] Garcia-Molina, H., Salem, K.: Sagas. In: Proceedings of the 1987 ACM SIGMOD international conference on Management of data, ACM Press (1987) 249–259

[8] Mukherjee, S.: A modified kangaroo model for long lived transactions over mobile networks. In: Proceedings from the 2002 WSEAS International Conference on E-Activities, Singapore (2002)

[9] Grace, P., Blair, G.S., Samuel, S.: A reflective framework for discovery and interaction in heterogeneous mobile environments. SIGMOBILE Mob. Comput. Commun. Rev. **9** (2005) 2–14

[10] Blair, G.S., Coulson, G., Robin, P., Papathomas, M.: An architecture for next generation middleware. In: Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing: Middleware'98, The Lake District, UK (1998)

[11] Karlsen, R.: An adaptive transactional system - framework and service synchronization. In: LNCS 2888, Springer–Verlag (2003)

[12] Karlsen, R., Jakobsen, A.B.A.: Transaction service management. an approach towards a reflective transaction service. In: 2nd International Workshop on Reflective and Adaptive Middleware, Rio de Janeiro, Brazil (2003)

[13] Arntsen, A.B., Karlsen, R.: Reflects; a flexible transaction service framework. In: Proceedings of the 4th International Workshop on Adaptative and Reflective Middleware (ARM05), Grenoble, France (2005)

[14] The Open Group: X/Open cae specification: The tx specification. (1995)

[15] Barga, R.S., Pu, C.: Reflection on a legacy transaction processing monitor. In: Proceedings of the Reflection '96 Conference, San Francisco, California, USA (1996)

[16] Wu, Z.: Reflective java and a reflective component-based transaction architecture. In: OOPSLA Workshop. (1998)

[17] Zarras, A., Issarny, V.: A framework for systematic synthesis of transactional middleware. In: Proceedings of Middleware'98. IFIP, Chapman-Hall (1998)

[18] Ramampiaro, H., Nygård, M.: CAGISTrans: providing adaptable transactional support for cooperative work. In: Proceedings of the 6th INFORMS conference on Information Systems and Technology (CIST2001), Springer–Verlag (2001)

[19] Prochazka, M.: Advanced transactions in enterprise java beans. Lecture Notes in Computer Science 1999 (2000)

[20] Chrysanthis, P.K., Ramamritham, K.: Synthesis of extended transaction models using ACTA. ACM Trans. Database Syst. **19** (1994) 450–491

[21] Cabrera, L., Copeland, G., Feingold, M., Freund, R., Freund, T., Joyce, S., et al.: Web services business activity framework (ws-businessactivity). ftp://www6.software.ibm.com/software/developer/library/WS-BusinessActivity.pdf (2005)

[22] OASIS: Business transaction protocol (btp). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=business-transaction (2002)

[23] Younas, M., Chao, K.M., Lo, C.C., Li, Y.: An efficient transaction commit protocol for composite web services. In: Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA 2006), 18–20 April, Vienna, Austria, IEEE Computer Society (2006)

[24] Mikalsen, T., Tai, S., Rouvellou, I.: Transactional attitudes: Reliable composition of autonomous web services. In: Workshop on Dependable Middleware-based Systems (WDMS), Washington d.c., USA (2002)