Disconnection Metadata for Distributed Applications in Mobile Environments

Nabil Kouici, Denis Conan, and Guy Bernard
GET / INT, CNRS Samovar
9 rue Charles Fourier, 91011 Évry, France
{Nabil.Kouici, Denis.Conan, Guy.Bernard}@int-evry.fr
Phone: 33 1 60 76 44 15, 45 34, 45 67   Fax: 33 1 60 76 47 80

Abstract

The need to continue to work in a mobile environment raises the problem of data availability in the presence of disconnections. Our approach aiming at solving this problem is to make a local replication of data and code on the mobile terminal. The system and the applications should also be reactive to mobile environment changes. The work presented in this paper is the continuation of Domint [6, 7], a platform to cope with disconnections in mobile environments for CORBA-based applications.

In this paper, we go further in the study of the role of disconnected entities by proposing meta-data to build patterns in order to (1) *choose which entities of the distributed application must become disconnected, and* (2) *state whether disconnected entities are necessary for the execution while being disconnected. We also outline the integration of these meta-data into the CORBA component-based architecture.*

Keywords: mobile computing, middleware, disconnection, meta-data, components.

1 Introduction

Last years have been marked by a rapid evolution in computer networks and machines used in distributed environments. We went from local area networks to large mobile networks, inter-connecting a wide variety of machines, including more and more mobile terminals like mobile phones and personal digital assistants. An important characteristic of mobile environments is that they suffer from frequent disconnections. A disconnection is a normal event in such environments and should not be considered as a failure. This has a profound impact on how applications are distributed. We distinguish between two kinds of disconnections: voluntary disconnections when the user decides to work on their own for saving battery or communication costs, or when radio transmissions are prohibited (as aboard a plane), and involuntary disconnections due to physical wireless communication breakdowns, such as in an uncovered area or when the user has moved out of the reach of a base station. We also consider the case where the communication is still possible but not at an optimal level, resulting from intermittent communication, low-bandwidth, high-latency, or expensive networks.

The need to continue to work in a mobile environment raises the problem of data availability in the presence of disconnections. The approach aiming at solving this problem is to make a local replication of data and code on the mobile terminal. The system and the applications should also be reactive to mobile environment changes. The work presented in this paper is the continuation of Domint [6, 7], a platform to cope with disconnections in mobile environments for CORBA-based applications. A proxy entity representing the remote server entity and called a disconnected entity, is deployed automatically on the client terminal. Connectivity management relies on an hysteresis mechanism to avoid too frequent state transfers and switchings between the disconnected and the remote server entities. For this purpose, we introduce a first-class partially connected mode (requests are sent to disconnected entities that forward them to remote server entities), in addition to the connected (requests are sent directly to remote server entities) and disconnected (requests are sent solely to disconnected entities that perform logging) modes.

In this paper, we go further in the study of the role of disconnected entities by proposing patterns in order to *(1)* choose which entities of the distributed application must become disconnected, *(2)* state whether disconnected entities are necessary for the execution while being disconnected, and *(3)* set priorities when the resource poverty or the connectivity don't allow all the necessary disconnected entities to be cached or updated on the mobile terminal. The remainder of the paper is structured as follows. Sections 2 and 3 present the motivations and the meta-data introduced to cope with these issues. Section 4 outlines the use of our meta-data in component-oriented middleware and gives a brief description of the implementation. Finally, Section 5 concludes.

2 Motivations and related work

A considerable effort of research is necessary in order to adapt distributed systems to the characteristics of mobile computing. Adaptation can be performed by the application (*laissez-faire* strategy), by the system (*transparent* strategy), or by both the application and the system (*collaboration* strategy)[26]. As surveyed in [13], there is much work dealing with mobile information access in client/server applications that demonstrates that the *laissez-faire* and the transparent approaches are not adequate.

Coda [16, 27, 26] is a file management system that defines the notion of implicit and explicit data. Implicit data are composed of the client history, thus reflecting recent accesses. Explicit data take the form of a data base, the *hoard data base*, built by the clients of the application, thus permitting non-optimal or bad choices when they don't fully understand the application. Consequently, applications may not be usable during disconnections. Odyssey [18, 19] adds data parameters: the *fidelity* and a *tolerance window*, well adapted to weak connectivity but not to disconnections. Rover [14] introduces two concepts: relocatable dynamic objects (RDO) and queued remote procedure call. Rover treats all the application objects in the same way; it doesn't take the application semantics into account and programmers must design and code their applications in terms of RDO. Therefore, as far as cache management for disconnection handling is concerned, the majority of the solutions don't associate the tagging of entities deployable on the mobile terminal with the prioritisation of the deployment.

The usage of meta-data has already been investigated in the field of reflective middleware in order to achieve adaptability and flexibility [3, 23]. XMIDDLE [4] defines meta-data gathered in an application profile describing how the middleware must behave when executing in a particular context. This profile contains information on the external resources (network bandwidth, battery resources...) and on the services offered by the middleware. Hence, it doesn't concern the application semantics. In software component technologies [28], extra functional properties are described in descriptors. For example, the CORBA component model (CCM) specification [20] proposes four types of descriptors: Software package descriptors, CORBA component descriptors, component assembly descriptors, and component property file descriptors. None of them treats the connectivity and the creation of the proxies on the mobile terminal: disconnection meta-data specification and description should be integrated into them.

The contribution of this paper is twofold. We propose three meta-data answering the following questions: Can this entity be used while being disconnected (disconnectability)? Is this entity necessary for the execution of the application in the disconnected mode (necessity)? Finally, is this entity having priority *w.r.t.* the synchronisation with the corresponding entity on the wired host (criticity)? This paper presents the first two properties that are static (considered already set when starting the application), the last one being dynamic (evolving during the execution) is left as an open issue. The second part of the contribution is the design of the corresponding patterns in the CORBA component model.

3 Meta-model for disconnection management

Some data, called meta-data, may be defined to be data which inform about the semantics of some other data and which allow their relevant use. In this section, we use meta-data to specify if an object can have a proxy on the mobile terminal (*cf.* Section 3.1) and if this proxy is necessary for the execution while being disconnected (*cf.* Section 3.2).

We use the word ”object” in its general meaning to define the granularity of the entities manipulated by the system (*e.g.*, a file in file-oriented systems, a record in data-oriented systems, an object in object-oriented systems, a component in the component-oriented systems). In addition, for the sake of clarity, we exemplify meta-data definitions with a wireless email browser application. This application is composed of three object types: `MailBoxManager` objects are responsible for creating, deleting, and localising `MailBox` objects; the latter objects provide methods for sending, receiving, and deleting messages; finally, `AddressBook` objects allow mailbox users to manipulate email addresses and mailing lists.

Object	Disconnectable	Necessity		
		Developer's choice	User's choice	Final
MailBox	yes	NO	–	NO
MailBoxManager	no	n/a	n/a	n/a
AddressBook	yes	UO	NO	NO

Table 1. Partitioning of objects of the wireless email browser example application: "UO", "NO", and "n/a" stands for "unnecessary object", "necessary object", and "not applicable", respectively.

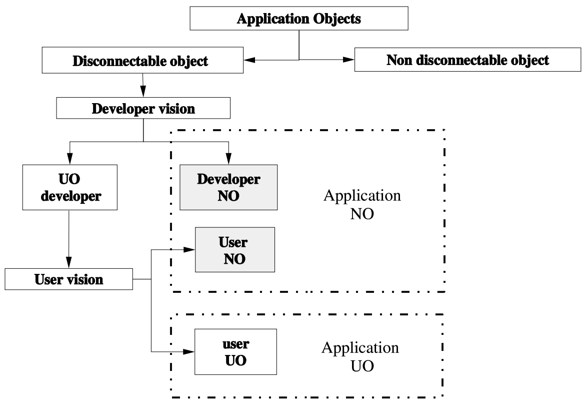

Figure 1. Meta-model of application objects' partitioning.

3.1 Disconnectability meta-data

The disconnectability meta-data contain information indicating whether a server object on the wired host can have a proxy on the mobile terminal. So, application objects are partitioned into two groups. Figure 1 depicts this partitioning plus others introduced by the necessity meta-data presented in Section 3.2. This choice is made by application developers[1] since they must build the application with these constraints. For example, for security reasons, application developers may decide to deploy some objects on dedicated secure hosts and to prevent clients from manipulating these objects on other hosts. Thus, proxies are not allowed for these objects and these objects are stated to be not disconnectable. In addition, application developers must design proxy version of disconnectable objects: as shown in [6, 7], proxy objects called disconnected objects are not the same as disconnectable server objects. Finally, disconnectable server objects must also be designed to enable future reconciliations with corresponding disconnected objects. In this paper, reconciliation issues are not discussed.

The second column of Table 1 exemplifies the assignment of the property "disconnectable" to objects of the wireless email browser application. *MailBoxManager* objects are prototypical objects that are not disconnectable for security reasons. Even if application developers allow these objects to have proxies on mobile terminals, let us consider the situation in which a user called the administrator creates `MailBox` objects for other users while the former is disconnected. The latter users can't use their mailboxes as long as the administrator hasn't reconciled by reconnecting. Therefore, the application is useless. *MailBoxManager* objects must then be classified so that being not accessible in disconnected mode, so not "disconnectable". In addition, concerning `MailBox` and `AddressBook` objects, they can have proxies in mobile terminals. While being disconnected, users can continue reading and writing their emails. They can access only the emails loaded before disconnections, newly arrived emails being kept on wired hosts `MailBox` objects and their new emails being effectively sent only when re-connecting. `AddressBook` objects are personal, not shared between users, and most of the time, passive objects. Hence, if users make changes on their `AddressBook` proxy, these operations don't impact other users' data, unless they decide to send their data, for example, to colleagues or friends. Soundly speaking, the latter operations imply that they are connected, not disconnected.

3.2 Necessity meta-data

The disconnectability meta-data presented in the previous section doesn't deal with the problem of multiple disconnected objects that must co-exist in the mobile terminal with reduced memory space. The necessity meta-

[1] In a component-oriented development process, as suggested by CCM, applications developers may be decomposed into component designers, composition designers, component implementers, component packagers, and component deployers.

data allows to classify the application objects in necessary (NO) and unnecessary objects (UO). A NO is a disconnectable object whose corresponding disconnected object on the mobile terminal must be present for the disconnected mode. Of course, the state of the disconnected object on the mobile terminal may not be up to date. On the contrary, the client part of the application may continue to work on the mobile terminal even if disconnected and without any disconnected objects (corresponding to UOs) being instantiated. Of course, the execution while being disconnected may not be equivalent to an execution while being connected or partially connected. This is acceptable provided that the connectivity information is visualised by an iconic image in the GUI. This is precisely the role of Domint to provide this connectivity information. Finally, we assume that applications are designed so that the cache of the mobile terminal can contain all the developers NOs.

As depicted in Figure 1, the partitioning of disconnectable objects in NO and UO is twofold: application developers but also users. Applications developers provide a first classification into “developer NOs” and “developer UOs” that can be, for “developer UOs”, changed by application users. Before launching the application, application users can force a developer UO to become necessary for their execution: this object is then called a “user NO”. Of course, application users can’t force a developer NO to become a user UO. Therefore, the set “application NOs” is the union of the sets of developer and user NOs.

The third column of Table 1 exemplifies the assignment of the property “necessity” to objects of the wireless email browser application. Clearly, `MailBox` objects are NOs. However, `AddressBook` objects are classified as being UOs by application developers because users can access their mailboxes and send messages to receivers whose addresses are known. In addition, when users start the wireless email browser application on their mobile terminals, they are informed that they can modify the necessity of `AddressBook` objects. To be more comfortable with their application, users can force the necessity of these object to “necessary”.

4 Design and implementation in CCM

After an overview of the CORBA CCM technology in Section 4.1, Sections 4.2 and 4.3 describe the disconnectability and necessity meta-data in CCM technology, respectively. Then, Section 4.4 gives the relations between those meta-data. Finally, Section 4.5 sketches the implementation.

4.1 Overview of the CORBA component model

Several industrial component models have emerged as a promising approach for building highly adaptable distributed applications. According to Szyperski [28], a component is defined as *“a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties”*. In the rest of this paper, we illustrate the instantiation of the meta-data in the CORBA Component Model (CCM) [20], for which open source implementations already exist: *e.g.*, OpenCCM [17] and micoCCM [24].

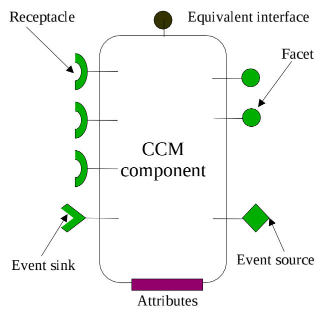

Figure 2. CCM component features.

In addition to be a promising specification, the choice of CCM is justified by the following arguments. Firstly, compared to other models such as EJB [9], COM [25], and .NET [8], CCM can be seen as an intersection of these models: compared to EJB, a CORBA component is created and managed by a home and executed in a container; compared to COM, a CORBA component offers and uses multiple interfaces, and allows introspection and navigation through interfaces; compared to .NET, a CORBA component can be written in several programming languages. Secondly, CCM is multi-language, multi-OS, multi-ORB, multi-vendor, contrary to EJB which is purely JAVA, and to COM and .NET which are purely Microsoft OS. Thirdly, CCM components can be segmented (see next paragraph).

A CCM component is programmatically characterised by a number of features (*cf.* figure 2): ports classified into synchronous (facets and receptacles) and asynchronous (event sources and event sinks), and attributes for component configuration. CCM components can be segmented (*cf.* figure 3). For each segment, CCM generates a separate skeleton. The segments are activated independently. Each segment has an independent state. If the container receives

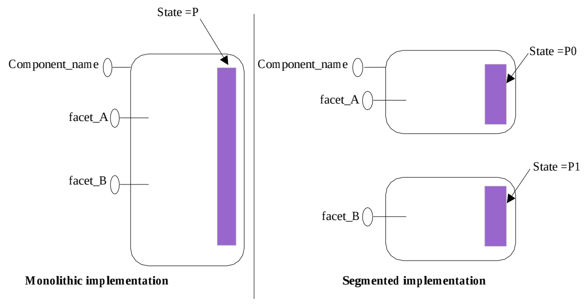

Figure 3. Monolithic and segmented components.

a request toward an interface of a component, the container activates only the segment which contains this facet. Each segment is separately identified. In the next sections, the concepts of disconnectability and necessity are applied at the two levels of granularity: component and segment.

4.2 Disconnectability meta-data in CCM

A disconnectable component is a component that can be instantiated as a proxy on the mobile terminal in order to be accessed while being disconnected. The proxy component called a "disconnected" component, contains only segments that can also be disconnected —*i.e.*, called disconnectable segments— and that are effectively disconnected —*i.e.*, called disconnected segments. In addition, a disconnectable component contains at least one disconnectable segment and if a component contains at least one disconnectable segment this component is disconnectable. Let $\mathcal{C}$ be the set of components, $\mathcal{S}_c$ be the set of segments of the component $c \in \mathcal{C}$, and $disc$ (*resp.* $diss$) be a predicate evaluating to true if a given component (*resp.* segment) is disconnectable, the previous statements are written as follows:

$$c \in \mathcal{C} : disc(c) \iff \exists s \in \mathcal{S}_c : diss(s) \qquad (1)$$

In the case of a monolithic component, all the features offered by the component (facets, receptacles, events, and attributes) are implemented in the same class. If it is disconnectable then the segment which corresponds to the totality of the component is disconnectable. In the case of segmented implementation, CCM defines a segment called the "main segment", which contains the features not explicitly serviced by the other segments declared by the component designer. In addition, the main segment has exclusiveness to handle the component context for introspection (search, navigation, connection...). In particular, it offers a localisation service of the other segments (*ExecutorLocator*). So, the disconnectability of a component implies the disconnectability of its main component. By a slight language abuse, the segment of a monolithic component will be called the main segment. The main segment of a segmented component $c \in \mathcal{C}$ is denoted $ms_c \in \mathcal{S}_c$. The previous statement together with equation 1 is written as follows:

$$c \in \mathcal{C} : disc(c) \iff ms_c \in \mathcal{S}_c : diss(ms_c) \qquad (2)$$

4.3 Necessity meta-data in CCM

By analogy with disconnectability, necessary components must have at least one necessary segment and if a component contains at least one necessary segment this component is necessary. We define another predicate $necc$ (*resp.* $necs$) evaluating to true if a given component (*resp.* segment) is necessary:

$$c \in \mathcal{C} : necc(c) \iff \exists s \in \mathcal{S}_c : necs(s) \qquad (3)$$

Also by analogy with disconnectability, the localisation service of the main segment is necessary to a disconnected component. Then, the necessity of a component implies the necessity of its main component:

$$c \in \mathcal{C} : nesc(c) \iff ms_c \in \mathcal{S}_c : necs(ms_c) \qquad (4)$$

4.4 Relation between disconnectability and necessity

It is clear that if a component or a segment is necessary, its presence in the mobile terminal is obligatory in order to use the application while being disconnected, thus it must be declared disconnectable. Therefore, we can say that the disconnectability meta-data is in sense "stronger" than the necessity meta-data since the user or the programmer must initially allot the disconnectability to segments and components then the necessity. However, the disconnectability of segments and components does not imply the necessity: the developer may let the component or the segment unnecessary and the user may override this choice (*cf* Section 3.2 and Table 1). The previous statements are written as follows:

$$c \in \mathcal{C} : nesc(c) \implies disc(c) \qquad (5)$$

$$c \in \mathcal{C}, s \in \mathcal{S}_c : ness(s) \implies disc(c) \qquad (6)$$

4.5 Implementation

In this section, we present different methods to specify the disconnectability and the necessity of components and segments in the OpenCCM platform [22].We have chosen OpenCCM because, to our knowledge, it is the sole CCM implementation which offers the specification of segmented components. In CCM, segmentation is specified in an extended OMG IDL language called CIDL (Component Implementation Declaration Language).

Figure 4 describes the CCM component-based software development process. The main activities of the CCM software process are the design, the implementation, the packaging, and the deployment. On this diagram, we trace out different solutions to integrate the disconnectability and necessity meta-data in CCM applications.

Our main objective is to describe the disconnectable/necessity segment/component to allow a transparent creation of the local copies of the segments/components and also to allow a transparent commutation between the copy in the server and the copy in the local host. We have chosen solution (1) because the other solutions (*(2)*, *(3)* and *(4)* in Figure 4) do not completely realise the objectives. Solution (*(2)* consists in adding new XML elements in the CORBA component descriptor. This solution requires the creation of a link between this descriptor and the client-side interception mechanism [21, 6, 7]. Therefore, in this case the interception service which allows to create the local copies and to make a switching is not transparent to the user. Solution (*(3)*: adding of a new descriptor, is similar to solution *(2)* and thus suffers the same inconveniences. Solution *(4)* consists in using aspect-oriented programming [15, 10] to integrate our meta-data in the the skeletons. The aspect-oriented programming is strongly dependent with the language of existing code and in our case the programmer must add for each segment the necessary code to describe the disconnectabable and the necessity meta-data.

The first solution noted *(1)* in Figure 4 consists in adding CIDL grammar rules to describe the disconnectability and the necessity of each component and each segment:

```
<executor_def>             ::=
   "manages" <identifier>
   [<executor_body>] ";"
<executor_body> ::
   "{" <body_def> "}"
<body_def>                 ::=
   <disconnection_management>
   <segment_def>
<disconnection_management> ::=
```

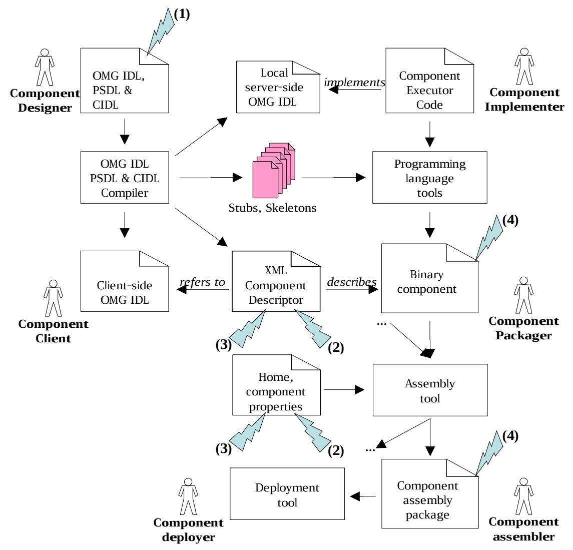

Figure 4. Development process of a CCM application.

```
   "disconnectable" "=" <value> ";"
   "necessity" "=" <value> ";"
<segment_def>              ::=
   "segment" <identifier>
   "{" <seg_member>+ "}"
<seg_member>               ::=
   <seg_per_decl> ";"
   <facet_decl>  ";"
   <disconnection_management>
```

We have added in the OpenCCM production chain [11] a rule to generate in each skeleton of segments our meta-data. The meta-data are used by the component home to add some information (disconnectability and necessity) in the segment IOR by tagging IOR profiles[2]. Meta-data added in IORs are used by the client-side interceptors for creating local proxies (disconnected components and segments) and transparently switching between disconnected local entities and remote server ones.

5 Conclusion

This paper has presented our work on the use of meta-data for managing caching in case of disconnections. The “disconnectable” property indicates if an entity can have a proxy on the mobile terminal and the “necessary” property

[2]In CORBA, the IOR interceptor mechanism [21] gives an opportunity to modify profiles of an IOR by adding “components” before it is published.

specifies if the presence of the proxy on the mobile terminal is mandatory for the execution of the mobile part of the application while being disconnected. These meta-data are filled by developers since disconnectable entities require specific design and implementation, and developers are in the best position to define the minimum set of necessary entities to be cached in the mobile terminal. In addition, users can supplement developers' choices to add more necessary entities when they want to be more comfortable with their applications. To this aim, meta-data have been added to a CCM version of Domint [6, 7].

The disconnectability and necessity meta-data don't cope with the priority between entities to be cached. For example, if a disconnection occurs and if all the necessary entities can't be created from or synchronised with the corresponding wired hosts entities, priorities of the hoarding or of the synchronisation must be specified in new meta-data called "criticity". Some other issues are user profiles for data recharging [5], replacement policies and incremental hoarding [1, 12, 2].

References

[1] H. Atzmon, R. Friedman, and R. Vitenberg. Replacement Policies for a Distributed Object Caching Service. In *International Symposium on Distributed Objects and Applications (DOA)*, pages 661–674, California, Irvine, USA, September 2002.

[2] A. Baggio. Replication and Caching Strategies in Cadmium. Technical Report 3409, INRIA France, Avril 1998.

[3] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas. An Architecture for Next Generation Middleware. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, London, 1998.

[4] L. Capra, W. Emmerich, and C. Mascolo. Exploiting Reflection and Metadata to built Mobile Computing Middleware. In *Workshop on Middleware Mobile Computing, IFIP/ACM*, Heidelberg, Germany, 2001.

[5] M. Cherniack, M.and Franklin and S. Zdonik. Expressing User Profiles for Data Recharging. *IEEE Personal Communications*, pages 32–38, Aug. 2001.

[6] D. Conan, S. Chabridon, O. Villin, and G. Bernard. Disconnected Operations in Mobile Environments. In *Proc. 2nd IPDPS Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, Ft. Lauderdale, USA, April 2002.

[7] D. Conan, S. Chabridon, O. Villin, G. Bernard, A. Kotchanov, and T. Saridakis. Handling Network Roaming and Long Disconnections at Middleware Level. In *Proc. Workshop on Software Infrastructures for Component-Based applications on Consumer Devices*, Lausanne, Switzerland, September 2002.

[8] M. Corporation. Microsoft developer network. http://www.msdn.microsoft.com.

[9] L. DeMichiel. *Enterprise JavaBeans Specifications, version 2.1, proposed final draft*. Sun Microsystems, http://java.sun.com/products/ejb/docs.html, Aug. 2002.

[10] T. Elrad, M. Aksit, G. Kicsales, K. Lieberherr, and H. Ossher. Discussing Aspects of AOP. *Communications of the ACM*, 44(10):33–38, Oct. 2001.

[11] A. Flissi. Inside OpenCCM. Technical report.

[12] A. Helal, A. Khushraj, and J. Zhang. Incremental Hoarding and Reintegration in Mobile Environments. In *Symposium on Applications and the Internet, SAINT*, Nara City, Nara, Japan, January 28 - February 01 2002.

[13] J. Jing, A. Helal, and A. Elmagarmid. Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, 31(2), June 1999.

[14] A. Joseph, J. Tauber, and M. Kaashoek. Mobile computing with the Rover toolkit. *ACM Transactions on Computers*, 46(3), 1997.

[15] G. Kiczales. Aspect-oriented programming. *ACM Computing Surveys*, 28(4), 1996.

[16] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. In *Proc 13th ACM Symposium on Operating Systems Principles*, number 5, pages 213–225, Pacific Grove, USA, 1991.

[17] R. Marvie and P. Merle. CORBA Component Model: Discussion and Use with OpenCCM. Technical report, Laboratoire d'Informatique Fondamentale de Lille, France, 2001. Submitted for publication.

[18] L. Mummert. *Exploiting Weak Connectivity in a Distributed File System*. PhD thesis, September 1996.

[19] B. D. Noble and M. Satyanarayanan. Experience with Adaptive Mobile Applications in Odyssey. *Mobile Networks and Applications*, 4(4):245–254, 1999.

[20] Object Management Group. CORBA Components. OMG Document formal/02-06-65, Version 3.0, June 2002.

[21] Object Management Group. Portable Interceptors. Interceptors Finalization Task Force. Published draft, Object Management Group, September 2001.

[22] ObjectWeb Open Source Software Community. OpenCCM home page. http://www.objectweb.org/openccm, 2003.

[23] N. Parlavantzas, G. Coulson, M. Clarke, and G. Blair. Towards a Reflective Component-based Middleware Architecture. In *Workshop on Reflection and Metalevel Architectures*, Sophia Antipolis and Cannes, France, June 13 2000.

[24] F. Pilhofer. Writing and Using CORBA Component. Technical report, ALCATEL, http://www.fpx.de/MicoCCM/, April 2002.

[25] D. Rogerson. *Inside COM*. Microsoft Press, 1997.

[26] M. Satyanarayanan. Fundamental Challenges in Mobile Computing. In *Proc 15th Symposium on Principles of Distributed Computing (PODC'96)*, pages 1–7, 1996.

[27] M. Satyanarayanan. Mobile Information Access. *IEEE Personal Communications*, 3(1), 1996.

[28] C. Szyperski, D. Gruntz, and S. Murer. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley, 2002.