Toward a Framework for Classifying Disconnected Operation Techniques

Marija Mikic-Rakic Nenad Medvidovic

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781 U.S.A.
{marija, neno}@usc.edu

Abstract

Distributed, decentralized, and mobile systems are highly dependent on the underlying network. Due to network connectivity failures, these systems must address the problem of disconnected operation, i.e., continued functioning in the absence or near-absence of network accessibility. A number of existing approaches provide support for disconnected operation by employing different techniques. What is currently missing, however, is a general understanding of the applicability of these techniques to different kinds of software systems, and the manner in which they affect the overall system dependability. This paper strives to improve that understanding. We present a framework for classifying disconnected operation solutions and assess several representative approaches according to the proposed classification. This study highlights several pertinent areas that are currently not supported, helping to motivate our future work.

1. Introduction

The emergence of mobile devices such as portable notebook computers, hand-held personal digital assistants (PDAs), and mobile phones, and the advent of the Internet and various wireless networking solutions make the computation possible anywhere. However, new challenges arise for software systems executing in such environments: they are becoming highly distributed, decentralized, and mobile, and therefore highly dependent on the underlying network. Unfortunately, network connectivity failures are not rare: mobile devices face frequent and unpredictable (involuntary) connectivity losses due to their constant location change and lack of wireless network coverage; the costs of wireless connectivity often induce user-initiated (voluntary) disconnection; and even the highly reliable WAN and LAN connectivity is unavailable between 1.5% and 3.3% of the time [25].

For this reason, network-dependent systems are challenged by the problem of *disconnected operation*, where the system must continue functioning in the (near-)absence of the network. Disconnected operation forces systems executing on each individual host to operate independently from other network hosts. This presents a major challenge for the software systems that are highly dependent on network connectivity, because each local subsystem is usually dependent on the availability of non-local resources. Lack of access to a remote resource can halt a particular subsystem or even make the entire system unusable.

There are several possible solutions to increasing the dependability of highly distributed and decentralized software systems in face of the connectivity losses:

- make remote data available locally,
- make remote code available locally,
- make remote dynamic system state available locally,
- reroute the communication in cases of partial disconnection from the network, and
- delay remote interactions until the connection is reestablished.

The goal behind each of these solutions is to temporarily mask the absence of connection by mimicking the system's continuous connectivity. The inconsistencies that may result from applying these solutions need to be resolved once the connection is re-established. Each of these solutions can be provided in a number of different ways, depending on the nature of the target application and on those aspects of the application's dependability that are of primary concern (e.g., availability, performance, scalability, security).

Most commonly used techniques for supporting disconnected operation are:

- *Caching* – locally storing remote data once it has been accessed in anticipation that it will be needed again [12],
- *Hoarding* – prefetching the likely needed remote data prior to disconnection [13],
- *Queueing remote procedure calls* – buffering remote, non-blocking requests and responses during disconnection and exchanging them upon reconnection [11],
- *Deployment and redeployment* – installing, updating, or relocating a distributed software system [1],
- *Replica reconciliation* – synchronizing the changes made during disconnection to different local copies of the same component [12],
- *Code mobility* – dynamic change of the bindings between code fragments and locations where they are executed [8].

What is currently missing, however, is a general understanding of the applicability of these techniques to different kinds of software systems, how and under what conditions they may be used (possibly in concert), how they affect the overall system dependability, and so forth. Also unclear is the applicability of these techniques in the growing class of architecture-centric, component-based software systems [18,19]. We believe that an understanding of these issues can help both to streamline existing and to develop future techniques in support of this area. This paper strives to improve that understanding. We present a framework for classifying disconnected operation solutions. We have performed an extensive study of existing techniques and identified a common set of criteria for their classification. In

turn, we have assessed several representative solutions according to the proposed classification. While this is still a work in progress, it has already clearly identified areas not addressed by current solutions. These areas, coupled with a study of the compatibility of different techniques, will frame our future research agenda.

The rest of the paper is organized as follows. Section 2 presents an overview of existing disconnected operation techniques. Section 3 describes our classification framework and assesses several representative solutions based on the identified criteria. The paper concludes with a discussion of open issues that will frame our future work.

2. Background

In this section we present an overview of existing disconnected operation solutions. They are organized according to the general approach they adopt.

2.1. Availability of data

2.1.1. Distributed file systems. Most of the early work on disconnected operation has been in the area of distributed file systems. Coda [12], Ficus [9], and D-NFS [6] have included extensive support for distributed file replication during disconnection and synchronization of replicas upon reconnection. These approaches use techniques such as caching and hoarding for file replication, and logging and version vectors for replica reconciliation. D-NFS and Coda also introduced the notion of an *agent*, which represents an intermediary between client and server components that handles their interaction during disconnection. An agent operates in two modes: connected and disconnected. In the connected state the agent forwards all of the client requests to the real server. This allows the agent to monitor client operations and prepare the application for disconnected operation. In the disconnected mode, the agent performs the necessary tasks (such as logging the client operations) needed to synchronize the replicas upon reconnection. While Coda and D-NFS focus on client-server applications, Ficus provides support for more general, peer-to-peer applications.

PFS [4] provides support for partially connected operation using a three-tier model, where an intermediary PFS host is inserted between a file server and a mobile client. PFS is a pseudo server for the mobile client and a pseudo client for the file server. This extra level of indirection enabled an efficient solution for partial disconnection, with a tolerable overhead during full connection. PFS provides a generic interface for application-directed adaptation to varying network quality of service requirements.

2.1.2. Distributed databases. In the area of distributed databases, disconnected operation has been addressed by approaches such as Thor [2], and Bayou [22].

In Thor, a relevant subset of database objects is cached prior to disconnection. Each transaction is logged during disconnection, but the clients can only perform "weak" transactions while disconnected. These transactions are committed only if they do not conflict with the transactions performed on the server.

Bayou is a platform of replicated, highly available, variable-consistency, mobile databases on which collaborative applications are built. Bayou focuses on providing application-specific conflict detection and resolution.

The main focus of both distributed file system and database approaches has been on supporting continuous availability of (passive) data only. Hence, they do not provide support for applications whose mode of operation during disconnection depends also on the availability of remote code and/or remote system state.

2.2. Availability of code

Providing continuous availability during disconnection by employing code mobility techniques has been the focus of approaches such as Rover [11], Jamp [23], Mobile extensions (ME) [3], Odyssey [17], and FarGo-DA [24].

The Rover toolkit provides two major programming abstractions: relocatable dynamic objects (RDOs) and queued remote procedure calls (QRPC). RDOs represent mobile code that can be dynamically loaded from a remote server and cached locally, while QRPC queues remote requests during disconnection and dispatches them upon reconnection. Rover uses version vectors to detect conflicts between different instances of a given RDO.

Jamp provides abstractions that support the migration of groups of objects and classes between nodes of the network. However, Jamp does not support object replication and, since mobile objects can only be in one location at a time, does not provide facilities for conflict resolution.

Mobile extensions (ME) provides location-independent, extensible facilities for deploying Web-based services. ME makes use of caching, hoarding, asynchronous messaging, and application-level adaptation to cope with network failures. This approach provides flexible and automatic resource management, since its employed techniques such as caching and hoarding can dramatically increase resource demands.

Odyssey is a set of extensions to the NetBSD operating system to support adaptation for a broad range of mobile information access applications. Odyssey provides monitoring of various system resources, notifies running applications of relevant changes, and enforces resource adaptation decisions. However, each application independently decides how to adapt to the notified change.

FarGo-DA is a programming model and a runtime infrastructure for automatic reconfiguration of an application during disconnection. FarGo-DA advocates disconnected operation awareness at system's design time. Additionally, since FarGo-DA is targeted at resource-constrained platforms, its main consideration is limited memory on such platforms. For this reason, FarGo-DA proposes the use of multi-modal components in which the developer specifies separate subcomponents to be used during connection and disconnection. Usually the subcomponent used during disconnection provides a subset of connected-mode functionality. Additionally, FarGo-DA assumes that the application developer provides the conflict resolution code as part of the multi-modal component.

It should be noted that none of the described code mobility techniques support automated selection of components that should be migrated, nor do they perform any analysis of the effects of code mobility on the running system.

2.3. Ad-hoc networking

In the area of ad-hoc networking, several approaches

have been proposed, including packet rerouting protocols (FORP [21]), predictive connection management with user input (PCP [14]), and adaptive wireless and mobile networking (Monarch [10]). These approaches are focusing on the mobility of the (human) user and are providing different techniques for rerouting when a mobile host changes location. However, they do not focus explicitly on disconnected operation, and therefore do not provide any support for the operation of a mobile client if it is completely disconnected from the network.

2.4. Other techniques

Some techniques have taken a more general approach to disconnected operation in which network disconnection represents only one point of failure in a given system. Failures of individual components are also examined and treated in a manner comparable to network failures. The goal of these techniques is to ensure that the system's operation will degrade gracefully in the face of failures. Replication is primarily used for increasing the reliability of individual components. An example such approach is RoSES [20], which provides a scalable framework for the analysis and design of system-wide graceful degradation.

3. Classification framework

3.1. Description

The overall structure of our proposed classification framework for disconnected operation approaches is organized around eight *categories*, as shown in Figure 1. Each category may have multiple *dimensions*, *subdimensions*, and *values*. The values are not necessarily mutually exclusive, meaning that a single approach may have zero, one, or multiple values corresponding to a given category, dimension, or subdimension. It should be noted that it was not our goal to identify all possible values in Figure 1 but rather to extract representative values from the existing approaches. We expect that the list of values will grow as we refine our classification framework. Missing approaches corresponding to values in Figure 1 indicate that no existing approach supports the corresponding property. Various other techniques exist that may be effectively applied in the context of disconnected operation (e.g., dynamic software architectures [5]). However, we feel that including such techniques in our framework at this time would be speculative as their effectiveness in this setting has not been demonstrated. In the remainder of the section we discuss each of the proposed classification categories in more detail.

3.1.1. Connectivity. Connectivity encompasses information about the nature of disconnection that a given approach supports (*type*) as well as how the *detection* of disconnection is achieved.

Connectivity type is further divided into two subdimensions: *predictability* and *degree* of connectivity. Predictability can have two values: *anticipated* and *sudden*. In cases of anticipated disconnection the system is aware that disconnection is going to occur, and usually can predict when it will happen [14]. In cases of sudden disconnection, the system is unaware of the disconnection beforehand.

We have identified the following values for the degree of connectivity: *total*, *partial*, and *low-bandwidth*. In cases of total disconnection, a given host is completely disconnected from the network. In cases of partial disconnection, the host is disconnected from the remote host with which it communicates, but there may be other hosts in the system to which this host is still connected, or can be connected. In cases of low-bandwidth connection, the host is connected, but through a low throughput connection. Low throughput connections necessitate the use of special techniques (e.g., compression) that would minimize the use of bandwidth.

Connectivity detection is further divided into two subdimensions: *accuracy* and *source*. Accuracy denotes whether disconnection is detected with no loss of data, by losing a single remote invocation (event), after which the subsystem recognizes that it has been disconnected and adjusts, or by losing multiple remote invocations. There are three possible sources of disconnection detection: (1) external agent, denoting a source (e.g., OS service) that is not a part of a given disconnected operation approach; (2) per host, denoting detection if an entire given host gets disconnected; and (3) per component, denoting detection of disconnection of individual software components.

3.1.2. Component types. The component types category includes information about the kinds of software components whose availability in the face of disconnection is supported by a given approach. This category is further divided into *active* and *passive* components. Active components can be *computation*, *communication*, *coordination*, or *interface* components, while passive components can be *files* or *dynamic data structures*.

3.1.3. Architecture. The approaches described in Section 2 use techniques such as component replication, migration, or network rerouting to increase the dependability of software systems during disconnection. However, these changes to a software system's architecture may have unforeseen effects on the running system. It is thus important to *analyze* the effects of the proposed changes prior to enacting them.

Static analysis may use (partial) architectural models to assess the validity of proposed run-time architectural changes prior to their deployment, possibly disallowing the changes. Dynamic analysis refers to the analysis performed after the deployment, of the effects of the performed modifications on the running target system. These techniques are described in more detail in [15].

In addition to the need for analyses, most of the approaches described in Section 2 impose certain architectural *topology* restrictions on the supported applications, such as client-server or peer-to peer. An important decision factor in determining the most suitable approach for a given system would be whether the system's topology is supported by the given approach.

3.1.4. Use of bandwidth. As outlined in the Introduction, partial or low bandwidth network connectivity is often present in highly mobile systems. In such cases, there is a critical need for efficient access mechanisms over networks with variable qualities of service. In our taxonomy, use of bandwidth is divided into two dimensions: *intelligence* and *efficiency*. Intelligence indicates whether a given approach uses an adaptive algorithm to optimize the use of bandwidth, while efficiency indicates whether the approach minimizes the use of bandwidth in cases of low bandwidth connection.

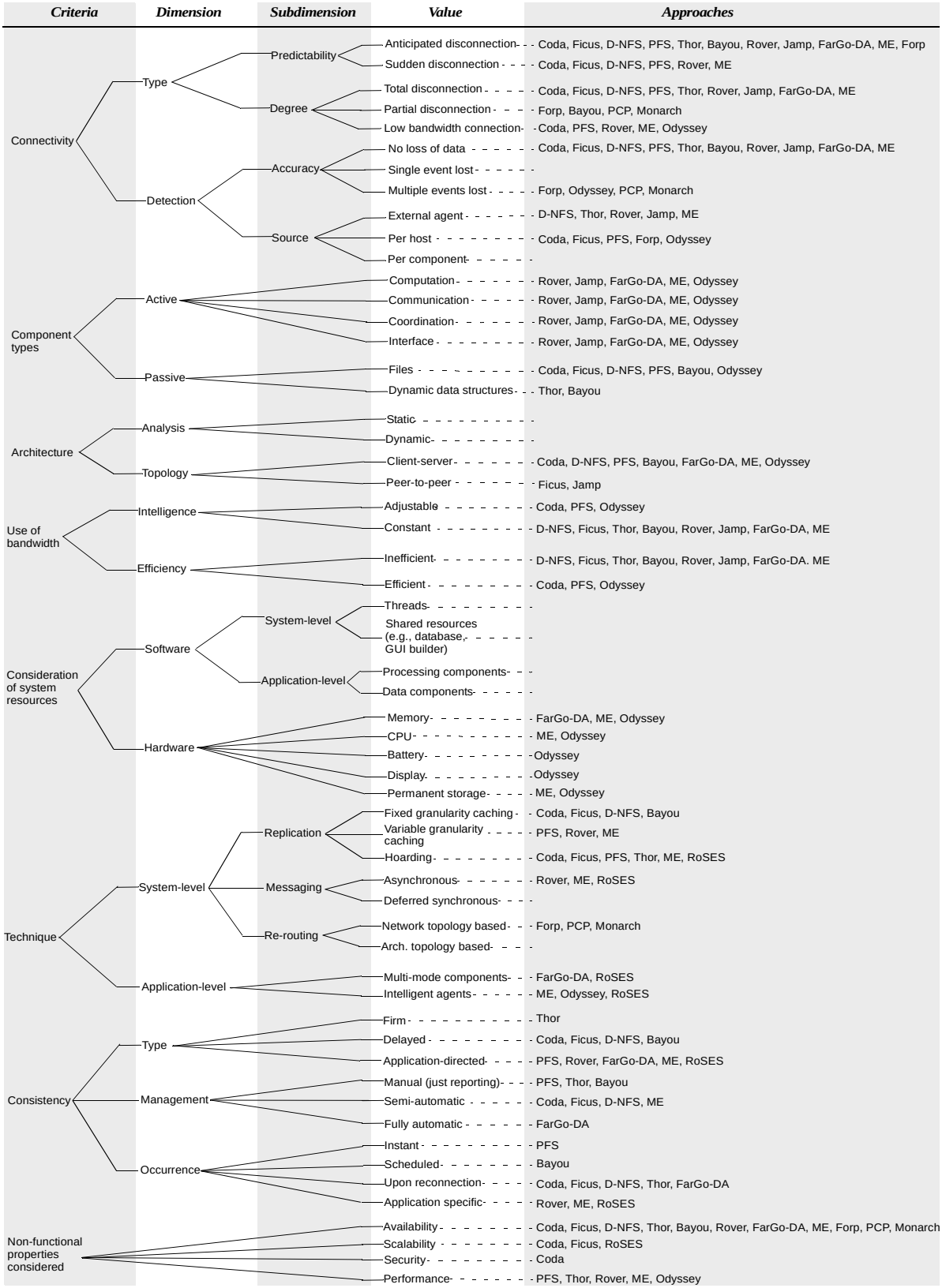

Figure 1. Classification Framework for Disconnected Operation Techniques.

3.1.5. Consideration of system resources. Dependability of a destabilized system is influenced by many factors. When selecting an approach that supports disconnected operation, it is important to know whether the approach considers the effects of the proposed changes on the system resources, and whether the available system resources on a set of affected hosts impose any restrictions on the proposed changes. If component migration is proposed by a given approach, it is important to assess whether the target device provides *hardware* resources (e.g., memory, CPU, display size) that the migrant component requires for normal operation. It is also important to assess the effects of *software* resources available on the target host (e.g., threads, existing processing components and their loads) on the migrant component. In our taxonomy, software resources are classified into *system-level* resources and *application-level* resources.

3.1.6. Technique. As outlined in the Introduction, there are a number of commonly used techniques for increasing system dependability during disconnection. We have classified these techniques into *system-level* and *application-level*.

System-level techniques are provided either at kernel- or middleware-level, and are further divided into *replication*, *messaging*, and *rerouting*. Replication subsumes techniques such as caching and hoarding, while messaging uses either asynchronous or deferred synchronous communication to delay remote interactions during disconnection. Synchronous messaging is not a feasible technique for supporting disconnected operation, since involved components would block for unpredictable periods of time.

Re-routing is a technique used to discover alternate paths of communication between mobile hosts. This subdimension can have two values: *network topology*, or *architecture topology* based rerouting. Network topology uses the information about the physical location of a given host and the network coverage of a given area. On the other hand, architecture topology uses additional information about the allowed communication paths among software components on each host to determine possible rerouting strategies. Both of these techniques can only support disconnected operation in cases of partial disconnection.

In the contrast to the above system-level techniques, several approaches [16,24] have proposed the use of application-level adaptation to increase system dependability during disconnection. For example, multi-modal components are designed with the a-priori knowledge that they may be executing in a disconnected mode. These components thus encapsulate two modes of operation: disconnected and connected. It is the responsibility of the application developer to design and implement a component's functionality such that it can be used during disconnection. The disconnected mode usually involves a subset of connected mode functionality, as well as methods for automated runtime conflict resolution.

Intelligent agents are special-purpose components whose role is to perform a set of activities which would translate a running application from a connected mode to a disconnected mode and vice versa.

3.1.7. Consistency. Several of the techniques described in Section 2 perform data, code, or system state replication to handle disconnected operation. Replication may require that changes made to different copies of a given component be synchronized upon reconnection. We have identified three dimensions of consistency: *type*, *management*, and *occurrence*.

Type denotes the extent to which the states of different replicas may diverge before they are synchronized. In *firm* consistency, the states of all replicas are always the same. This is achieved by either disallowing the updates to different replicas or by performing simultaneous, blocking updates to all replicas. In *delayed* consistency, replicas can be in different states, and consistency management (i.e., replica reconciliation) is performed upon reconnection. Some approaches also allow *application-directed* type of consistency, i.e., specification of (possibly different) consistency types for each application-specific operation.

Consistency management denotes the manner in which the reconciliation is performed. Some approaches *just report* inconsistencies, which are then resolved by the application user. In *semi-automatic* management, some conflicts are resolved automatically, while others are reported to the user for (manual) resolution. Finally, in *fully automatic* management all conflicts are resolved automatically, without the user's involvement.

Occurrence denotes the time at which the reconciliation is performed. In cases of *instant* reconciliation, each update will result in an immediate attempt to reconcile all replicas. In cases of *scheduled* updates, updates are planned and performed according to some, usually component-specific algorithm. Most of the existing approaches perform reconciliation *upon reconnection*. Finally, some approaches provide *application-directed* scheduling of reconciliation.

3.1.8. Non-functional properties. Existing disconnected operation approaches have considered different non-functional properties in the interest of increasing system dependability. Most commonly considered non-functional properties are availability, performance, security, and scalability. There are other relevant non-functional properties such as safety, reliability, utility, and so on. However, no existing disconnected operation approaches have focused on these properties.

3.2. Assessment of existing approaches

We have classified a number of representative approaches using our framework. The results of this classification are shown in the right-most column in Figure 1. In the remainder of this section we discuss these results.

Most of the existing approaches focus on anticipated disconnection, and on maximizing the system's availability during disconnection. Coda, PFS, ME, and Rover support both anticipated and sudden disconnection and provide support for low-bandwidth connection. With the exception of Bayou, partial disconnection is supported only by ad-hoc networking approaches (recall Section 2.3).

Coda, PFS, and Odyssey make intelligent and efficient use of the network bandwidth. However, none of the remaining approaches adjust their operation for a low-bandwidth connection. Instead, they assume either fully connected or disconnected mode of operation.

With the exception of Fargo-DA (for memory), only ME and Odyssey take into consideration system resources (CPU, disk space, battery, and so on). These approaches recognize that a given mobile host will not have unlimited resources to fully support techniques such as hoarding, and that certain trade-offs have to be made (e.g., providing continuous availability of only the most frequently used com-

ponents).

Fargo-DA, ME, and Odyssey use application-level disconnected operation techniques, while the remaining approaches leverage different combinations of system-level techniques. The most commonly used system-level technique is (some form of) replication, while the approaches that employ delayed communication via messaging only use asynchronous messaging.

Finally, none of the studied approaches perform any kind of analysis of the effects of the changes on the running system. They also fail to take into consideration other software resources (e.g., number of threads), or perform architecture-based re-routing (recall Section 3.1).

4. Conclusion

In this paper we have presented an attempt at classifying the existing disconnected operation approaches. A general understanding of these approaches and the techniques they employ is needed to effectively support system dependability in the face of disconnection. The existing approaches attack the problem of disconnected operation from four general perspectives: data housed in (1) static files and (2) dynamic data structures, and functionality implemented in both (3) inactive and (4) active software components. Typically, an approach will focus on a specific subset of these four categories (e.g., support for off-line access to static files only). Our classification framework is a step in the direction of understanding the (in)compatibilities among the existing techniques and suggesting the best possible approach or combination of approaches (e.g., coupling a passive file-based approach and an active component-based approach) for the problem at hand.

This work is preliminary and much remains to be done. A natural next step is to gain further experience by evaluating additional known disconnected operation approaches using the framework outlined in this paper. Such an evaluation will, in turn, be used to fine-tune the framework itself. In addition, we plan to study the compatibilities of the different criteria, dimensions, subdimensions, and values, which would help with identifying techniques that can be used in concert. Finally, further study of existing disconnected operation techniques will highlight additional areas that are currently not supported, which would motivate and help to streamline our future work.

5. References

[1] A. Carzaniga et. al. A Characterization Framework for Software Deployment Technologies. *Technical Report*, Dept. of Computer Science, University of Colorado, 1998.

[2] S. Chang and D. Curtis. An Approach to Disconnected Operation in an Object-Oriented Database. *3rd International Conference on Mobile Data Management*, January 2002, Singapore.

[3] M. Dahlin, B. Chandra, L. Gao, A. Khoja, A. Nayate, A. Razzaq, A. Sewani. Using Mobile Extensions to Support Disconnected Services. *University of Texas Department of Computer Sciences Tech Report TR-2000-20*, June 2000.

[4] D. Dwyer and V. Bharghavan. A Mobility-Aware File System for Partially Connected Operation. In *ACM Operating Systems Review*, Vol. 31, No. 1, Jan. 1997, pp. 24-30.

[5] Dynamic Software Architectures Resources. http://www.ics.uci.edu/~peymano/dynamic-arch/

[6] M. E. Fiuczynski and D. Grove. A Programming Methodology for Disconnected Operation. *Technical Report*, University of Washington, March 1994.

[7] K. Froese and R. Bunt. Scheduling Write Backs for Weakly Connected Mobile Clients. In *Proc. of the 10th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Palma de Mallorca, Sept. 1998.

[8] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Trans. on Software Engineering*, May 1998.

[9] J. S. Heidemann et al., Primarily Disconnected Operation: Experiences with Ficus. *Second Workshop on Management of Replicated Data*. IEEE, November 1992.

[10] D. B. Johnson and D. A. Maltz. Protocols for adaptive wireless and mobile networking. *IEEE Personal Communications*, 3(1), February 1996.

[11] A. D. Joseph, A. F. de Lespinasse, J. A. Tauber, D. K. Gifford, M. F. Kaashoek, Rover: a toolkit for mobile information access, *Proceedings of the fifteenth ACM symposium on Operating systems principles*, December 1995, Colorado.

[12] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, vol. 10, no. 1, February 1992.

[13] G. H. Kuenning and G. J. Popek. Automated Hoarding for Mobile Computers. Proceedings of the *16th ACM Symposium on Operating Systems Principles, (SOSP-16)* St. Malo, France, October 5-8, 1997.

[14] M. Madi, P. Graham, and K. Barker. Mobile Computing: Predictive Connection Management With User Input. *Technical Report*. Dept. of Computer Science, Univ. of Manitoba, 1997.

[15] M. Mikic-Rakic and N. Medvidovic. Architecture-Level Support for Software Component Deployment in Resource Constrained Environments. *First International IFIP/ACM Working Conference on Component Deployment*. Berlin, June 2002.

[16] W. Nace and P. Koopman. A Product Family Approach to Graceful Degradation. In *Proceedings of International Workshop on Distributed and Parallel Embedded Systems*, Germany, October 2000.

[17] B. Noble, et. al. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, St. Malo, France, October 1997.

[18] D.E. Perry, and A.L. Wolf. Foundations for the Study of Software Architectures. *Software Engineering Notes*, Oct. 1992.

[19] M. Shaw and D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. *Prentice-Hall*, 1996.

[20] C. Shelton, et al. A Framework for Scalable Analysis and Design of System-Wide Graceful Degradation in Distributed Embedded Systems. *WORDS 2003*, January 2003.

[21] W. Su and M. Gerla. IpV6 Flow Handoff in Ad-Hoc Wireless Networks Using Mobility Prediction. *Proceedings of IEEE Global Communications Conference*, pp 271-275, Rio de Janeiro, Brazil, December 1999.

[22] D. B. Terry, K. Petersen, M. J. Spreitzer, and M. M. Theimer.The Case for Non-transparent Replication: Examples from Bayou. *IEEE Data Engineering*, December 1998.

[23] M. T. Valente, R. Bigonha, M. Bigonha and A. Loureiro. Disconnected Operation in a Mobile Computation System. *Workshop on Software Engineering and Mobility*, Toronto, Canada, May 2001.

[24] Y. Weinsberg, I. Ben-Shaul. A Programming Model and System Support for Disconnected-Aware Applications on Resource-Constrained Devices. *International Conference on Software Engineering 2002*, Orlando, Florida, May 2002.

[25] Y. Zhang, V. Paxon, and S. Shenkar. The Stationarity of Internet Path Properties: Routing, Loss, and Throughput. *Technical Report*, AT&T Center for Internet Research at ICSI, May 2000.