PERSISTENT CONNECTIVITY AND RELIABLE MESSAGE DELIVERY
IN COMPONENT-BASED MOBILE APPLICATIONS

RADU LITIU and ATUL PRAKASH
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109-2122, USA
Email: {radu.aprakash}@eecs.umich.edu

Abstract
Applications for mobile and wireless environments will need to support computing devices with a wide range of capabilities, varying network connectivity, and increasing mobility of users. We have designed a component-based framework for building adaptive distributed applications. Distributed applications are viewed as consisting of connected components that typically implement data streaming, processing, and filtering functions. Our framework provides mechanisms for run-time reconfiguration of applications to allow them to adapt to the changing operating environments. Components or whole applications can be moved to different hosts during execution, while maintaining seamless communication connectivity with other components. This paper presents techniques used to implement component mobility, ensure application consistency and reliable message delivery during component moves, as well as maintain persistent connectivity during transient network failures. We show potential benefits of maintaining seamless connectivity between mobile components.

Keywords: mobile users, mobile applications, dynamic application reconfiguration, persistent connectivity, application parking

1. Introduction

Mobile computing environments are characterized by a wide variety of computing devices, ranging from high-end machines to simple devices such as personal digital assistants (PDAs) or cell phones. Network connectivity can be highly dependent on the carrier’s coverage, device capability in terms of power and hardware, and the location of the user. In this paper, we consider support for situations in which mobile users are running applications that are part of a distributed computing environment, being connected to services, data sources, and to groupware applications of other users. Unlike current environments, we consider new demands on the infrastructure where a user needs to move his applications, or their components, from one computing device to another (e.g., from a PDA to a desktop), while maintaining connectivity with other applications. It would be desirable to provide support to users so that they are able to move applications across devices while maintaining seamless communication connectivity with other applications (Figure 1).

Operating in a mobile environment also raises the problem of dealing with the inherent unreliability of mobile network connections and variations in connection quality. For many applications, it would be desirable that the underlying middleware layers mask transient network and communication failures.

Mobile IP [7] addresses the problem of maintaining connectivity between moving hosts. As a computer moves from one wireless network to another, the computer’s home IP address is maintained. When only some applications move across hosts, moving the IP address along with those applications would not be appropriate, since there might be other applications running on that machine that rely on the same mobile IP address.

To address the above issues, we have designed and implemented (in Java) a mobile component framework, called DACIA (Dynamic Adjustment of Component InterActions), for building and executing reconfigurable distributed applications. The flexible architecture proposed allows modular applications to dynamically change their structure and relocate components, while maintaining persistent connectivity with other components. Logical connections between components are maintained even when underlying TCP/IP connectivity is lost for short times (as it can happen in today’s GPRS networks) or the device’s IP address changes due to mobility.

Our work complements existing solutions proposed to address the variability and resource constraints encountered both in wired and mobile computing systems, such as Daedalus/BARWAN [4], on-demand dynamic distillation tech-
Relocate components
Manage application structure
Handle message exchange
Handle component connectivity
Maintain connections between hosts

MONITOR 1
ENGINE 1
MONITOR 2
ENGINE 2
ENGINE 3
HOST 3 - PDA
HOST 1 - fixed PC
HOST 2 - mobile PC

Figure 2. A DACIA distributed application is a directed graph of connected components (ovals represent components). An engine runs on every host. It manages the local components and the connections between components, both local and across different hosts. The monitor gathers performance data and implements application-specific relocation and reconfiguration policies.

niques [2], Odyssey [6], Conductor [10], and CANS [3]. It addresses some of the complex software engineering issues identified in supporting a high degree of flexibility in ubiquitous and pervasive computing [1, 8, 9]. We address part of that problem, so that distributed application components can be dynamically relocated without losing connectivity to other components. They can potentially be programmed to relocate according to a specified policy, such as moving with the location of the user, or to optimize the usage of currently available resources.

The rest of the paper presents an overview of the architecture of the system and outlines algorithms that implement component mobility and ensure application consistency and message integrity during component moves. We also show potential benefits of maintaining persistent connectivity between mobile components.

2. DACIA Architecture

DACIA is a framework for building adaptive distributed applications in a modular fashion, through the flexible composition of software modules implementing individual functions. A DACIA application can be seen as a directed graph of connected components. The links between components indicate the direction of the data flow within the application.

In DACIA, a component is a PROC (Processing and ROuting Component). A PROC can apply some transformations to one or multiple input data streams. It can synchronize input data streams; it can split the items in an input data stream and send them alternately to multiple destinations. A PROC is identified system-wide using a unique identifier.

The engine decouples an application and component-specific code and functionality from the general administrative tasks such as maintaining the list of PROCs and their connections, migrating PROCs, establishing and maintaining connections between hosts, and communicating between hosts. A DACIA distributed application (Figure 2) uses an engine on every host it runs on. We chose to use an engine per application per host, as opposed to sharing an engine running on a host between multiple applications, in order to minimize the cost of communication between PROCs and between PROCs and the engine.

PROCs communicate by exchanging messages through ports (Figure 3). Communication can be either synchronous or asynchronous. In the case of asynchronous communication, the messages received by a PROC are inserted into the PROC’s message queue. Every PROC has a thread that handles the messages in the queue, usually in FIFO order. We designed and implemented the communication mechanisms in DACIA with the goal of minimizing the overhead of message exchange. DACIA provides a lightweight solution to local communication, by co-locating local PROCs within the same address space. Thus, message exchange translates into simple procedure calls.

The engine maps virtual connections between PROCs to either local or remote network connections, and handles data transfers accordingly. Multiple virtual remote connections between pairs of PROCs are multiplexed over a single network connection between two engines. The connectivity between remote PROCs is maintained as long as the corresponding engines are connected. Sharing physical connections reduces the cost of establishing network connections in a highly dynamic application, where PROCs often connect to each other or they are disconnected.

The engine of an adaptive application may work in conjunction with a monitor. While the engine provides the mechanisms used to build and reconfigure applications, the monitor implements the policy layer in a DACIA application. The engine is not aware about the semantics of the application. The monitor keeps track of the application performance, makes reconfiguration decisions, and instructs the engine accordingly. The engine is responsible for establishing and removing con-
connections between components and for moving components to other hosts.

An application can be dynamically reconfigured by reordering or relocating some components, or replacing a set of components with a different set of components, possibly connected in a different configuration. Components can be connected or disconnected. New components can be dynamically loaded, and existing components can be removed or relocated to different hosts. The performance of the application can be improved by changing the way different application components interact and their location of execution, thus taking advantage of the resources available system-wide. Our solution to runtime reconfiguration ensures that the application executes correctly both during and after the reconfiguration.

3. Component Mobility

One of the key features of our architecture is the ability to move components between hosts. Our work addresses the problem of capturing the state of a component and restoring it at the destination. To reduce the overheads of component movement, thus the amount of data that has to be moved, we do not transfer the execution state of a PROC (e.g., program counter, stack and registers content, thread state, etc). However, a moving PROC carries with it the state of its data members, the messages received and not handled yet, and the state of its connections. PROC migration happens at well-defined times with respect to the execution of the PROC. Before a PROC moves, if a message is currently being handled, the handling routine completes. When a PROC moves to another host, all messages left in its message queue move with the PROC.

A major challenge in implementing PROC mobility in DACIA has been to make the move transparent to communicating PROCs, allow other PROCs to send messages to a moving PROC, and ensure that these messages are delivered reliably to their destination. The FIFO order of message delivery and processing at the destination has to be preserved as components relocate to a new engine. Message exchange has to be synchronized with a PROC move.

The PROC is locked while it is moving, thus preventing other PROCs from sending messages to it. The message send operation blocks while the PROC is moving. The sender PROC (the corresponding engine) retrieves a *remoteProc* reference corresponding to the receiver, then sends the message to the new instance of the receiving PROC at its new location. We avoid acquiring a lock corresponding to the receiving PROC every time a message is sent. A *location* flag is set prior to a move and reset after the move completes. The flag is tested prior to the sender acquiring a lock, thus eliminating the need for a lock if the receiver is not undergoing a move. At the same time, a counter indicating the number of outstanding messages prevents a PROC move operation from being executed while the PROC is receiving and handling a message from another PROC.

At the destination host, a locking mechanism synchronizes the PROC initialization and the updates to the engine data structures with the ongoing message send operations addressed to the moving PROC, and with potential subsequent move operations that may be initiated before the data structures corresponding to the moving PROC are updated. We use message sequence numbers and a sliding window protocol to prevent messages potentially originating at different hosts from being delivered out of order to the moving PROC.

The destination engine sends notifications about the PROC move to all other engines that have PROCs connected to the moving PROC. Upon receiving a move notification, an engine updates its local information about the PROC’s location, thus avoiding the need for message forwarding in the future. A notification contains the ID of the moving PROC, the new location of the PROC, and the change sequence number that keeps track of the order of changes applied to the PROC (a change can be a move, connect or disconnect). The sequence number is used to prevent old PROC move notifications received out of order from being applied.

4. Persistent Connectivity

In many cases, the temporary failure of a connection between engines can be made transparent to PROCs and applications. When a network connection is broken, the engines will try to re-establish the connection during a timeout interval. The timeout interval can be set on an application basis, regardless of the physical network settings. Assuming that the disconnection is temporary, an engine caches messages addressed to a remote PROC until the connection is re-established. The use of a timeout for re-connection allows an administrator to briefly shut down an application running on a host, and immediately
restart it, without other connected applications noticing it. All inter-component connections are transparently re-established, and neither one of the applications loses any state information. We have found this feature useful during the testing, debugging, and upgrading of distributed applications.

A logical connection between PROCs is maintained even if the underlying physical connection changes. PROCs can move between hosts while maintaining persistent connectivity to other PROCs. The structure of the application does not change and the flow of data in the system is not interrupted. Using move notification information, the engines transparently re-establish inter-PROC connections over the physical connections between the hosts of the communicating PROCs. The seamless connectivity between DACIA components offers a great benefit to mobile users, who can move applications from one device to another without having to manually re-establish all connections to other parties. It can also provide transparency of the location of PROCs and users, if so desired.

Through application parking [5] (Figure 4), component mobility and persistent connectivity can be used to support the offline operation of interactive applications. Initially developed in the context of groupware applications, application parking is suitable to any interactive distributed application. Using DACIA, a parked application can continue to maintain state and to participate, on a limited basis, to collaborations on the user’s behalf, while the user is disconnected or is not active. When the user reconnects, eventually from a different place, she can take over the control from the parked application.

![Figure 4. Application parking. (a.) In traditional groupware applications, when a user disconnects, its state has to be saved on a server. If the user later connects to a different host, the state is transferred between servers and between the new server and client. (b.) Using DACIA, the user can park her client to a fixed, connected host. While the user is disconnected, a parked client can continue to maintain state and its connections, and it can interact with collaborative partners.](image)

5. Conclusions

Our work addresses the challenges of using reconfigurable component-based applications to support the specific needs of mobile and wireless environments. Solutions exist for maintaining connectivity between mobile hosts, that attach to different networks and change their IP addresses. However, maintaining logical connectivity between mobile applications has not been thoroughly investigated.

In this paper, we propose a solution for implementing component mobility so as to ensure reliable message delivery among components in the presence of mobility. Our design also provides support for persistent component connectivity when TCP/IP connectivity incurs transient failures or the IP address of the device changes. To ensure that messages are reliably and orderly delivered both during and after a component relocation, component moves are synchronized with message delivery. We discuss the applicability of component mobility and persistent connectivity to supporting off-line operation of interactive applications.

We are currently extending DACIA to address the security concerns of both mobile components and the hosts where they execute.

Acknowledgments

This work has been supported in part by the National Science Foundation under Grant No. ITR 0082851.

References


