

Middleware support for component-based ubiquitous and mobile computing applications

Davy Preuveneers, Peter Rigole, Yves Vandewoude, Yolande Berbers
Department of Computer Science
Celestijnenlaan 200A
B-3001 Leuven, Belgium
{davy,peterri,yvesv,yolande}@cs.kuleuven.be

ABSTRACT

The pervasive computing paradigm promises support for user mobility and personalization of services. People will be shifting from the desktop to more resource constrained devices, such as personal handheld devices and smartphones, to consume these services. However, developing and deploying mobile and pervasive services for a broad range of systems with different capabilities and limitations is a daunting task without proper middleware support. At Middleware 2005, we will demonstrate Draco, our Java-based component middleware, running unmodified on both high-end and low-end devices, featuring support for resource discovery, service adaptation and service mobility at runtime. The demo will also show the use of context information for automated service reconfiguration and development support for component-based applications.

1. INTRODUCTION

The growing presence of mobile devices, such as laptops, PDAs and smartphones, along with advances in wireless network communication technologies, have created new opportunities for making the applications more intelligent and supportive to the user. Intelligent applications are able to adapt themselves to the context of the user and the new device whenever they move from one host to another in a mobile computing environment. In the following sections we will discuss how our component middleware Draco is capable of providing the necessary infrastructure to support the ubiquitous and mobile computing paradigm. The following real life conferencing scenario provides a nice show case of a context-aware mobile service and of what our component middleware is capable of:

Mr. Smith is a sales representative of a large company and is attending a business meeting where the economic analysis manager is presenting the results of a recent survey administered by their company and where the participants are discussing marketing strategies. All attendants can use a

display to interact with the shared whiteboard in the conference room. Mr. Smith has to leave the meeting early as he is scheduled to visit a client to address some long-standing maintenance problems. During the trip, he would like to stay in touch with his colleagues and discuss the proposals being presented. The conference room allows for broadcast meetings for audience members who cannot contribute live to the meeting, but who are able to watch and listen to the presentation remotely. They can interact during the meeting using a chat application to send and respond to questions or to read the text that the presenter enters. Upon detection that Mr. Smith is leaving, the conferencing client moves from his display in the conference room to his personal wireless handheld device. As the I/O capabilities of a handheld are rather limited, some parts of the conferencing client can later on move to other devices in order to save the battery from draining, to provide a larger display, more bandwidth or processing power whenever appropriate devices show up in the vicinity of Mr. Smith.

Some challenges that need to be addressed:

1. The mobile conferencing client should be deployable on different hardware platforms, ranging from desktop systems to mobile handheld devices, without any modification or manual reconfiguration of the service.
2. The resource requirements of the conferencing client should be clearly and unambiguously specified to make sure that the application can be deployed within the available resources of a particular device whenever it moves.
3. The state of the conferencing client should be preserved while moving from one system to another to ensure that the contact list, the current conversation, the presenters notes and the slides are not lost during transfer.
4. The supporting infrastructure should be aware of Mr. Smith leaving the conference room and initiate the relocation of the conference client to preferably his personal handheld device or otherwise to an appropriate device in his immediate vicinity and that remains in his vicinity.
5. Optimizing resource usage in a mobile environment requires on the fly discovery of devices and resources in

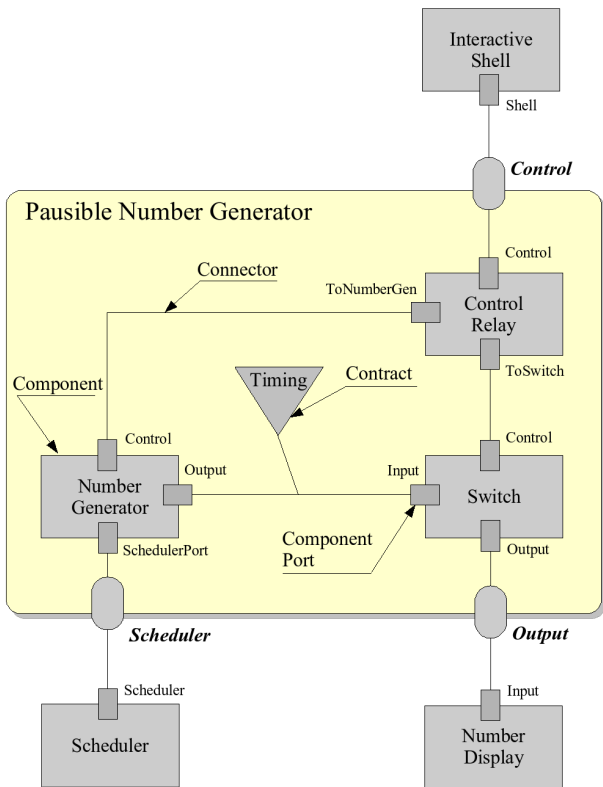


Figure 1: A simple component-based application

an ad-hoc network with the ability to check whether these devices provide the required capabilities.

These actions should happen at runtime, i.e. the client service is active and should remain active with as little user intervention and interruption as possible. It is clear that reaching this stage of non-intrusive automated deployment and configuration requires proper middleware support.

2. DRACO: OUR MIDDLEWARE FOR COMPONENT BASED APPLICATIONS

The current trend towards context-awareness is explained by the growing need for services that are more sensitive to user requirements but less dependent on user attention [1, 2]. Hence, a critical success factor of a middleware platform in a mobile and ubiquitous computing environment is its support for adapting services automatically to a broad range of hardware – such as PDAs, smartphones and laptops – and available resources. For this reason, we use *components* as modular building blocks for the design and deployment of adaptable services. Our component-based development approach [4] provides flexible adaptation capabilities within applications to optimize resource usage and to adapt to changing working conditions. The innate property of a component is its black box nature providing an encapsulation of the implementation and its message-oriented communication mechanism. This allows us to instantiate an application with only these components that are able to run decently given the resource constraints of a specific device. A simple component-based application is shown in Figure 1.

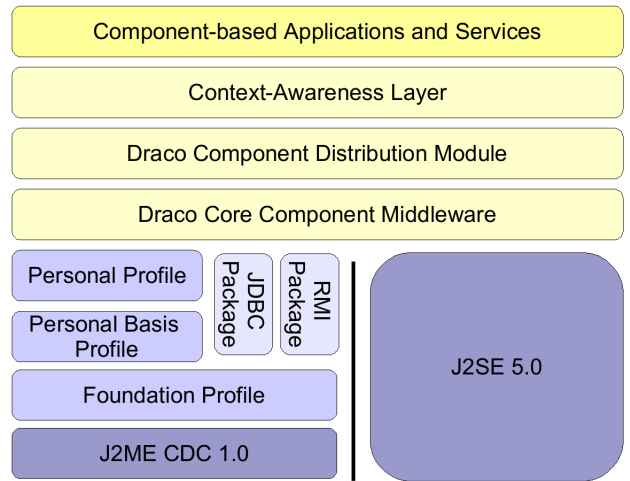


Figure 2: Draco running on multiple Java VMs.

Components provide the functional building blocks of a service and use *Component Ports* as communication gateways to other components. *Connectors* serve as the message channel between these ports. Communication between component ports is managed by sending asynchronous *Messages*. *Contracts* [7] define restrictions or requirements on two or more components or ports, for example, to limit or guarantee memory availability or to define timing constraints. *Composite Components* are prefabricated compositions of component instances and act to the outside as regular components. Their component ports are exported internal component ports. The example in Figure 1 shows how three components, *Number Generator*, *Switch* and *Control Relay*, are composed into a new composite component *Pausible Number Generator*. The number generator repeatedly provides random numbers with a user-defined frequency. If the *Switch* component is enabled, then the numbers are forwarded to the *Number Display* component to show them on a screen. The random number generator interacts with the *Scheduler* component to get notified of when to send out a new number. The user is able to interact with the application by using the *Interactive Shell* component. It provides a prompt to send messages to the application, for example, to pause and later continue the random number generation by (de)activating the switch. These messages are intercepted by the *Control Relay* component, which forwards the messages to the right component. The *Timing* contract specifies timing constraints for sending messages from the number generator to the switch to ensure that messages are delivered on time, and is only shown here for demonstration purposes.

Draco, our Java-based component middleware, is an extensible runtime system for components designed to be deployed on embedded devices. It acts as a component container that instantiates components and manages all connections between components. Draco has support for non-functional concerns, such as the possibility of live updating with better suited components [5], negotiating and monitoring resource contracts [7], distributing the execution and relocating component-based applications [3]. A simplified overview

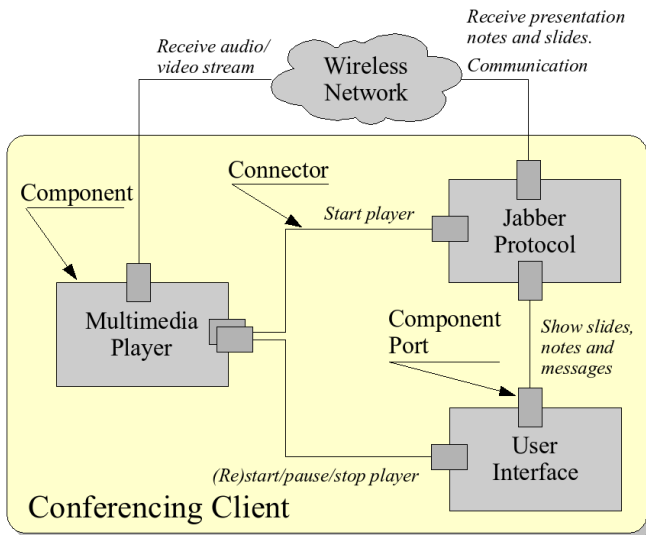


Figure 3: Building blocks of the component-based conferencing client

of our infrastructure for component-based mobile services is given in Figure 2. It shows the full stack of our component middleware including the virtual machine (VM). We are able to run the same compiled version of a component, using the J2ME Personal Profile 1.0 (J2ME PP) virtual machine on the handheld and a Java 2 Standard Edition 5.0 (J2SE) on the desktop. J2ME PP is based on the Connected Device Configuration (CDC) and supports resource-constrained devices with a GUI toolkit based on AWT and requires only 1MB of RAM (not including application requirements). We currently use the IBM J9 Personal Profile implementation for PocketPC 2003 on the Qtek 9090. The component middleware Draco [6] runs unmodified on top of both virtual machines and is able to deploy the same precompiled components, which are packaged as a Java archives (JAR), on both environments as long as these components adhere to the API supported by both VMs.

3. DEVELOPMENT SUPPORT

Furthermore, Draco provides special language support to facilitate the development of component-based applications. Several extra keywords have been added to the Java language to model components, ports and messages. Figure 4 shows the ubiquitous 'Hello World' example implemented in the Draco component language. The Draco preprocessor will translate such component specifications to plain Java. We are currently working towards integration of our tools within the Eclipse environment (see Figure 6) to further ease the process of developing component-based applications.

For more information about the development of Draco, we refer interested readers to the following website which provides a downloadable version of Draco to experiment with: <http://www.cs.kuleuven.be/~yvesv/?q=Draco>

4. THE DEMO

The demo we will show at Middleware 2005 closely follows the scenario briefly discussed in the introduction:

```

package components.HelloWorld;

import draco.Draco;

dracomponent HelloWorld
{
    multicastport OutgoingMessage 4 {
        outmessage Goodbey;
        outmessage Stop;
    }

    portgroup MessageBox 8 {
        outmessage Hello;
    }

    inmessage Start {
        Draco.getShell().writeln("I received the 'Start'-message",
            HelloWorld.this);

        newmessage x Hello;
        x::greet = "Hello World!";
        MessageBox..x;
    }

    inmessage Stop {
        Draco.getShell().writeln("I received the 'Stop'-message",
            HelloWorld.this);

        newmessage x Goodbey;
        x::leave = "Goodbey, cruel world!";
        OutgoingMessage..x;
    }

    inmessage Hello {
        System.out.println($$inmessage::greet);
    }

    inmessage Goodbey {
        System.out.println($$inmessage::leave);
    }
}

```

Figure 4: Example 'Hello World' component implementation

1. The Draco middleware is running on a laptop and on several handheld devices (Qtek 9090). All devices are able to communicate using the wireless network. Each device is carrying its own hardware and resource description. This information will allow us to adapt applications to optimize resource usage on a given host.
2. The laptop acts as a server hosting a repository of components and multimedia files being used during the demo. The handheld devices are only running the Draco middleware. The server also provides the application model of the instant messaging client, specifying all mandatory, optional and alternative components. The application model is a loosely coupled composition of components providing support for jabber-based communication, slide shows, audio and video transmissions.
3. The personal handheld device is going to upload its device description to the server and download the components of the instant messaging application that fit and run properly on the given device, and thus leaving out all optional heavyweight components.
4. Audio and/or video will be enabled at runtime on the handheld device depending on the presence of available resources (memory, bandwidth and processing power) or other changing working conditions (battery).

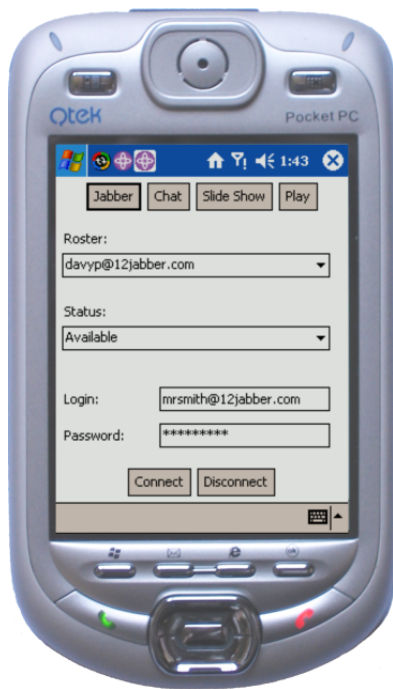


Figure 5: Screenshot of the component-based conferencing client running on a PDA

5. Using the quality of the WiFi signal, location awareness will be demonstrated with a particular component moving from one host to another to follow a mobile user, while preserving the state of the component.
6. The device will discover in a peer-to-peer fashion the presence of resources in its neighborhood and relocate components within the composition of the client application to the new computing environment while the application remains active. As such, one single application is being executed on multiple devices to optimize resource usage and quality of service.
7. The application will again be reconfigured given the new computing environment. Without resource limitations, all components in the application model will be enabled.

5. REFERENCES

- [1] D. Preuveneers and Y. Berbers. Adaptive context management using a component-based approach. In *Proceedings of 5th IFIP International Conference on Distributed Applications and Interoperable Systems*, volume 3543 of *LNCS*, pages 14–26. Springer, June 2005.
- [2] D. Preuveneers and Y. Berbers. Automated context-driven composition of pervasive services to alleviate non-functional concerns. In *Proceedings of the 2nd International Workshop on Software Aspects of Context (IWSAC'05)*, pages 28–38, Santorini/Greece, July 2005.
- [3] P. Rigole, Y. Berbers, and T. Holvoet. Mobile adaptive tasks guided by resource contracts. In *the 2nd*

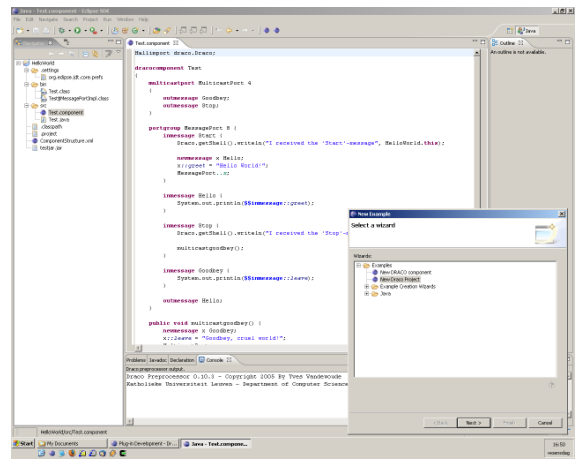


Figure 6: Eclipse support for Draco

Workshop on Middleware for Pervasive and Ad-Hoc Computing, pages 117–120, Toronto, Ontario, Canada, October 2004.

- [4] D. Urting, S. Van Baelen, T. Holvoet, and Y. Berbers. Embedded software development: Components and contracts. In *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems*, pages 685–690, 2001.
- [5] Y. Vandewoude and Y. Berbers. Run-time evolution for embedded component-oriented systems. In B. Werner, editor, *Proceedings of the International Conference on Software Maintenance*, pages 242–245, Canada, October 2002. IEEE Computer Society.
- [6] Y. Vandewoude, P. Rigole, D. Urting, and Y. Berbers. Draco : An adaptive runtime environment for components. Technical Report CW372, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, December 2003.
- [7] A. Wils, J. Gorinsek, S. Van Baelen, Y. Berbers, and K. De Vlaminc. Flexible Component Contracts for Local Resource Awareness. In C. Bryce and G. Czajkowski, editors, *ECOOOP 2003 Workshop on resource aware computing*, July 2003.