Configurable Services for Mobile Users

Andreas Rasche and Andreas Polze
Humboldt University of Berlin
10099 Berlin, Germany
{rasche|apolze}@informatik.hu-berlin.de

Abstract

Mobile devices, such as cellular phones, personal digital assistents (PDAs), and organizers, are becoming increasingly popular. Due to the high volatility of those devices, the achievable quality-of-service (QoS) for mobile services can hardly be predicted. Even for one particular type of device - say a PDA - the implementation of a mobile service may use different communication interfaces over time (i.e.; wireless LAN, IrDA).

Within this paper, we present a new approach towards configuration of component-based services for mobile systems. Starting from a XML-based configuration language, which defines a set of rules for component configuration depending on a number of environmental parameters, our approach allows for instantiation and configuration of components.

In contrast to many other approaches targeting distributed multimedia-style application on PC-class computers, our framework focuses on the extension of distributed services onto mobile devices. As proof-of-concept scenario we have implemented a configurable distributed video surveillance application on the basis of the Microsoft Distributed Component Object Model on Windows 2000 and on the Windows CE-based Pocket PC platform. [1]

1 Introduction

Mobile devices, such as cellular phones, personal digital assistents (PDAs), and organizers, are becoming increasingly popular. The raw number of those devices is expected to exceed the number of personal computers (PCs) by an order of magnitude within the next couple of years. More and more of those devices have powerful communication interfaces which lend naturally to the construction of ad hoc networks and allow for the implementation of mobile services.

[1] This work has been partially sponsored through a research grant from Microsoft Research Cambridge.

Due to the high volatility of mobile devices, the achievable quality-of-service (QoS) for mobile services can hardly be predicted. Even for one particular type of device - say a PDA - the implementation of a mobile service may use different communication interfaces over time (i.e.; wireless LAN, IrDA). Those interfaces may provide different network QoS (bandwidth, latency, jitter, etc.). This dilemma is depicted in Figure 1.

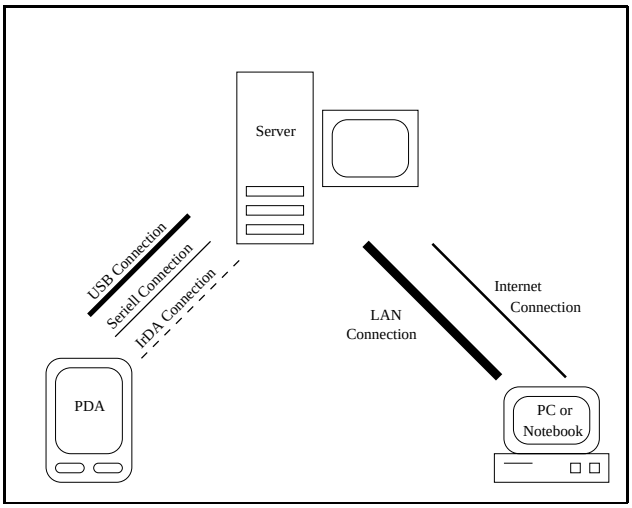

Figure 1. Problem - Services dealing with varying system properties

Classical approaches towards device-specific services for mobile users differentiate the type of quality of service offered by the type of the endsystem (i.e.; ASP.NET produces different formats of web-content for a cell phone than for a Pocket PC than for a PC-class computer). However, most of those approaches do not take into account the change of available QoS (which might be location-specific) when talking to the same mobile device.

Within this paper, we present a new approach towards configuration of component-based services for mobile systems. Starting from a XML-based configuration language,

which defines a set of rules for component configuration depending on a number of environmental parameters, our approach allows for instantiation and configuration of DCOM (Distributed Component Object Model) components. Those components may be reconfigured or replaced during runtime of the system, once the available network QoS changes.

In contrast to many other approaches targeting distributed multimedia-style application on PC-class computers, our framework focuses on the extension of distributed services onto mobile devices. For our case studies, we have decided to use Microsoftś DCOM and the Windows CE-based Pockect PC as implementation platform. This platform is unique in the sense that it extends the reach of the DCOM middleware down to the area of embedded and mobile devices. However, results of our research are applicable to other platforms as well.

The remainder of the paper is structured as follows: Section 2 briefly discusses the basis of our work - the idea of Aspect-Oriented Programming (AOP) and the usability of XML (eXtensible Markup Language) as an aspect language. Section 3 presents configuration as an aspect in distributed systems and describes our approach towards an aspect language for (dynamic) (re-) configuration description. Section 4 discusses implementation issues, whereas Section 5 presents our proof-of-concept scenario - a mobile video surveillance application. Related work is discussed in Section 6. Section 7 concludes the paper.

2 Aspect-Oriented Programming and XML

Object-oriented and component-based programming are focusing on the functional interfaces of objects or components. A variety of programming languages and component frameworks have established different mean for the description of function signatures, parameter types and return values as well as data values exposed by objects or components.

However, these classical approaches fall short in describing non-functional component properties, such as memory usage, component location, or migration. Many of those non-functional properties are cross-cutting in the sense that they are influenced by multiple components simultaneously. Object-oriented structuring concepts, such as data encapsulation, classes, inheritance and polymorphism, are not applicable to those non-functional properties. Rather than being localized in an object or a class, those properties typically express themselves by little code fragments here and there - a phenomenon called code tangling.

The idea of aspect-oriented programming ([6]) consists of the introduction of special structuring mechanisms for non-functional component properties. Those structuring mechanisms can be implemented as extension of a programming language - or, as a completely separate decription language which stands beside the implementation language. Tools (so-called aspect weavers) allow for automatic interconnection of aspect description and component implementation during the build- and deployment process (One may actually see the DCOM interface definition language - IDL - as a special aspect language addressing component distribution. Its corresponding aspect weaver is the IDL compiler - *MIDL.exe*).

Although, from an academic standpoint, the extension of a programming language is always interesting, educating and fun to do, we opt for separating aspect descriptions and implementation language. Because of its widely available tool support and its generality, we have decided to use the eXtensible Markup Language (XML) as language for describing the aspect of component (re-) configuration.

XML [13] uses document type definitions (DTDs) to define rules for the structuring of documents. Within a DTD, user-definable *tag* and *attribute* constructs can be declared. XML documents using those tags and attributes can be automatically checked for adherence to a given DTD by an XML parser. The extensibility of XML allows for easy creation of problem-specific (aspect-) languages. Although XML documents are somewhat cryptic, they are still human readable.

3 Configuration as an Aspect in Distributed Systems

The configuration problem for distributed systems consists of component enumeration, instantiation, and placement of components, and identification of interconnections among components. An actual configuration may be applicable for a given set of environmental parameters (constraints), such as computing nodes, types of interconnects, spare CPU capacity, memory space, or communication bandwidth. A configuration description may contain multiple configurations (enumeration of components and interconnections) which are applicable under certain sets of constraints given by the environment. A rule set may be used to select an actual configuration from a configuration description. Rules may take the current setting of environmental parameters as measured by *observers* (probes) as input parameters.

In our approach, a configuration description document consists of three major sections: The *configuration section* lists participating components with their attributes and their connectors. The *observer section* defines a set of available observers and the type of parameters they deliver. A set of rules defining the configurations which are applicable under given environmental constraints are specified in the *profile section*.

In our approach, components are implemented following the Microsoft Distributed Component Object Model (DCOM). Connections are represented by so-called *connectors* which implement specific communication protocols. These connectors can be as simple as a TCP/IP socket or a shared memory interconnect. However, connectors may rely on device-specific APIs, such as the Remote API (RAPI) available with Windows CE.

The principle layout of a configuration section is described in Figure 2.

```
<!ELEMENT configuration (component+,connector+)>
<!ATTLIST configuration configurationname CDATA #REQUIRED>

<!ELEMENT component (attribute+,input+,output+)>
<!ATTLIST component componentname CDATA #REQUIRED>
<!ATTLIST component UUID CDATA #REQUIRED>
<!ATTLIST component location CDATA #IMPLIED>

<!ELEMENT attribute EMPTY>
<!ATTLIST attribute attributename CDATA #REQUIRED>
<!ATTLIST attribute type CDATA #REQUIRED>
<!ATTLIST attribute value CDATA #REQUIRED>

<!ELEMENT input EMPTY>
<!ATTLIST input inputname CDATA #REQUIRED>

<!ELEMENT output EMPTY>
<!ATTLIST output outputname CDATA #REQUIRED>

<!ELEMENT connector EMPTY>
<!ATTLIST connector type CDATA #REQUIRED>
<!ATTLIST connector inputcomponent IDREF #REQUIRED>
<!ATTLIST connector inputmethod IDREF #REQUIRED>
<!ATTLIST connector outputcomponent IDREF #REQUIRED>
<!ATTLIST connector outputmethod IDREF #REQUIRED>
```

Figure 2. Configuration section of a configuration description DTD

Within the configuration section, components and connectors are enumerated. A component is identified by a name, a location to be loaded in, and a UUID (unique 128-bit identifier) as used to identify COM+/DCOM components. Furthermore, a component may have attributes. These attributes carry a name, a type and a value. Attribute types at this point are based on the DCOM type system. Finally, input and output links for components are defined. Connectors connect to these input and output links; they also have a type.

To adapt application behavior to the underlying hardware properties a profile is needed which selects an applicable configuration of a service according to a given set of system properties. These properties are measured by observers. The *configuration manager* collects the values of all observers, compares them to the configuration description document, and selects an applicable service configuration for the current situation.

The observer section of a configuration description document lists available observer components. Each observer is assigned a name. The type of data measured by the observer is specified as well as the observerś location. Special pseudo-locations are being used to describe the co-location of observers with either client or server components for a distributed client/server application. An excerpt from the DTD showing the observer section is listed in Figure 3.

```
<!ELEMENT observer EMPTY>
<!ATTLIST observer observername CDATA # REQUIRED>
<!ATTLIST observer UUID CDATA # REQUIRED>
<!ATTLIST observer location CDATA #IMPLIED>
<!ATTLIST observer type CDATA #REQUIRED>
```

Figure 3. Observer section of a configuration description DTD

The profile section (see Figure 4) of a configuration description document associates certain outputs (property values) of observer components with configurations enumerated in the configuration section. A profile associates property settings and configurations by defining lower and upper bound for property values. An example for an XML configuration description adhering to the DTD discussed in Figures 2 - 3 is presented in Figure 7.

```
<!ELEMENT profile (property+)>
<!ATTLIST profile name CDATA #REQUIRED>
<!ATTLIST profile configurationname IDREF #REQUIRED>

<!ELEMENT property EMPTY>
<!ATTLIST property observername IDREF #REQUIRED>
<!ATTLIST property minvalue CDATA #REQUIRED>
<!ATTLIST property maxvalue CDATA #REQUIRED>
<!ATTLIST property type CDATA #REQUIRED>
```

Figure 4. Profile section of a configuration description DTD

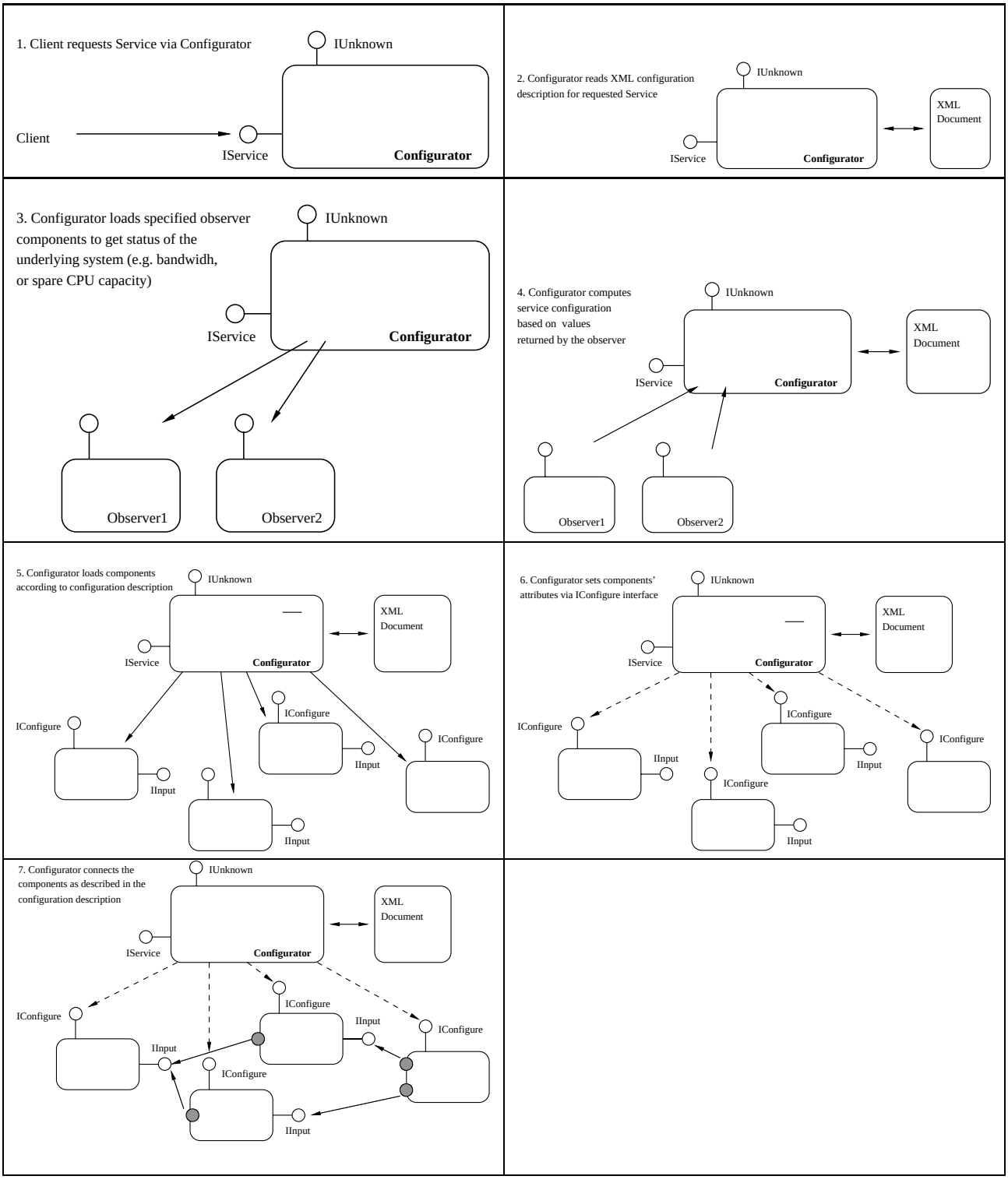

Figure 5. Steps of an actual configuration process

4 Implementation Issues

4.1 Configuration Manager

The central component of our framework is the *configuration manager*. Its primary tasks are measuring of system properties, loading corresponding components, and connection establishment among components.

As already mentioned, system properties are measured by observer components. Observers can be simple functions with a pre-defined signature or DCOM components implementing the *IObserver* interface. The *IObserver* interface provides a *GetValue* method, which delivers the value of a system property.

At system configuration time, the configuration manager determines the current values of all properties enumerated in the XML configuration description and selects the configuration to be loaded. This process has several steps as depicted in Figure 5.

4.2 Components of a Configurable Service

Each component of a configurable service has to implement the interface *IConfigure*. The configuration manager uses this interface to set componentsáttributes (via *SetAttribute*) and to establish connections among components (via *Connect_XXX*).

The *Connect_XXX* methods do not only represent the type of the connector but also implement some communication details. To illustrate that point let us consider a RPC-connector. The *Connect_RPC* method takes as parameter a reference to another component. This parameter is stored inside the connector component and used throughout lifetime of the service.

Our framework comes with a variety of different connectors and corresponding *Connect_XXX* methods in the *IConfigure* interface. However, the framework is extensible and it is relatively easy to integrate methods for new connectors such as the method *Connect_TCP*, which would implement communication via TCP/IP sockets.

4.3 Pre-configuration phase

The biggest challenge in the configuration process is the specification of fine-grained configuration rule sets. During a pre-configuration phase different configurations of the service are tested off-line. Measurements obtained during this phase form the basis for specification of profiles. Currently, these measurements are carried out manually. However, a tool which automates off-line testing is under development.

A similar approach towards automatic configuration of distributed systems has been described by Chang et al. [4]. In this work, a performance database is used to store characteristics and behavior of different configurations off-line. This database is consulted at application runtime.

5 Demo Scenario - a Mobile Video Surveillance Application

5.1 Microsofts Windows CE 3.0

As a proof-of-concept scenario, we have implemented a mobile video surveillance application on Compaq iPAQ 3630 Pocket PCs running Windows CE 3.0 and on desktop PCs running Windows 2000. Windows CE was released 1996 by Microsoft [12]. It is an operating system for embedded devices. Windows CE supports a subset of the Win32-API, an application programming interface for Windows 9X/NT/2000. The development process for Windows CE programs is host-based and uses cross-compilers which generate executables for the actual embedded device. From a programmerś perspective, the Windows CE operating system is a similar target as a desktop PC running Windows 2000. However, there are some important exceptions.

5.2 Communication and Windows CE 3.0

Windows CE 3.0 provides a variety of communication options. Windows CE offers support for serial connections through the Win32 Serial API. Furthermore, Windows CE supports the Network Driver Interface Specification (NDIS) for local area networks, and the Infrared Data Association (IrDA) standard for wireless communication.

Besides these low-level communication interfaces, Windows CE provides TCP/IP and the Winsocket-API, and a the host-based Remote-API (RAPI). However, what makes Windows CE special among embedded operating systems is its support for the Distributed Component Object Model (DCOM). This extends the reach of standard middleware down to the embedded devices. Our framework for configurable services takes advantage of those advanced capabilities of Windows CE and provides connectors for TCP/IP, RAPI and DCOM.

The Remote-API (RAPI) implements a lightweight non-standard remote procedure call (RPC). A server process on Windows CE-side receives calls from the CeRapiInvoke-method from PC-side. With the Remote-API, data from and to the Windows CE device may be transfered in either block or stream mode. We have implemented connectors using both communication modes.

For our scenario, we assume a security guard monitoring a bank. Cameras mounted throughout the building deliver video streams to a base station (desktop PC). Motion detection algorithms can be used to extract scene shifts from those video streams. The security guard is walking through

the bank. He carries a mobile digital assistant which she may plug into base stations located throughout the building to communicate with the video surveillance service. Communication can be established using serial connection, infrared or wireless LAN.

Because of resource constraints (CPU power, network bandwidth, memory), it is not possible to display real time video streams directly on the Pocket PC device. However, depending on network bandwidth, the device may well be able to display still images Depending on configuration, the video surveillance service may convert video streams into compressed pictures. Worst case scenario is the transmission of text notices about scene shifts (at the highest compression ratio no images are transmitted). In that case the guard has to check recorded video data at the desktop PC to figure out the nature of those scene shifts.

The general configuration rule ensures better quality of the delivered information with higher network bandwidth and higher CPU power of the clientś device.

5.3 Components of the Demo Application

In our demonstration application, the video stream originates from the program *LkDemoView* which is was adapted from the *Intel Open Source Vision Library* [5]. The modified program sends pictures via DCOM from a Webcam to a server process - the *data collector*. *LkDemoView* also contains a motion tracker component, that analyzes the video stream and generates text information. Further components are a JPEG-filter with the attribute *compression rate* as well as player components that show pictures or text.

5.4 Pre-configuration

There are several possible configurations for our proof-of-concept scenario. With a powerful LAN connection it is possible to transmit video streams with high quality. Figure 6 illustrates this scenario.

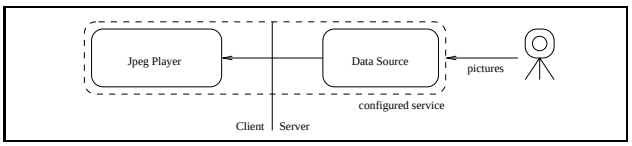

Figure 6. service configuration for a PC with LAN connection

The scenario depicted in Figure 6 involves 2 desktop PCs running a data collector component (*data source*) and a jpeg player. Both components communicate via DCOM RPC.

Figure 7 lists the configuration description document for this application scenario containing only one possible configuration called *highband*. Within this document, two observers are enumerated. The profile section of the document contains a single rule which specifies CPU and bandwidth constraints under which the configuration *highband* is applicable.

In order to establish the configuration *highband*, components *data collector* and *JpegPlayer* have to be instantiated on nodes localhost and client, respectively. Besides an output link delivering jpeg-encoded video data, the *data collector* also provides textual output. Both components are to be interconnected via DCOM RPC. In our case, the input method *jplayerinput* is interconnected with the output method *jpegout*.

```
<?xml version="1.0"?>
<configurationdescription name="mobile video surveillance">

<observer observername="CPU_capacity" location="Client" type="int"
                UUID="{924C6EAB-2F7B-45B3-9EC2-093E3A444033}"/>
<observer observername="bandwidth" location="Client" type="int"
                UUID="{5547EF58-9C62-4B58-992E-D33299E2E064}"/>

<profile name="LANPC" configurationname="highband">
        <property observername="bandwidth" minvalue="0" maxvalue="400"
                type="observer"/>
        <property observername="CPU_capacity" minvalue="0" maxvalue="1000"
                type="observer"/>
</profile>

<configuration configurationname="highband">
        <component componentname="data collector" location="localhost"
                UUID="{987D78F8-D08C-4615-99AF-C235E06984C5}">
                <attribute attributename="informationfreq" type="int"
                        value="clientparameter"/>
                <output outputname="jpegout"/>
                <output outputname="textout"/>
        </component>
        <component componentname="JpegPlayer" location="client"
                UUID="{4982F0E0-070F- 40A4-952D-27C4C9C29954}">
                <input inputname="jplayerinput"/>
        </component>
        <connector type="RPC"
                inputcomponent="JpegPlayer" inputmethod="jplayerinput"
                outputcomponent="data collector" outputmethod="jpegout"/>
</configuration>
</configurationdescription>
```

Figure 7. Configuration description for high-bandwidth video transmission

Figure 8 shows a configuration for a mobile PDA with a serial connection. The service delivers compressed pictures and also transmits text at scene shifts to ensure timely reaction of our fictious security guard.

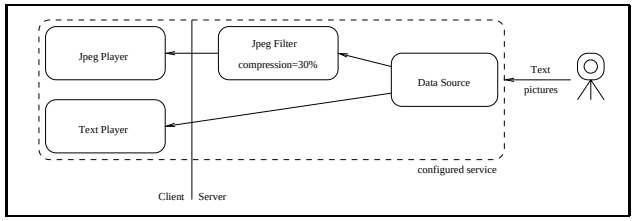

Figure 8. service configuation for a PocketPC with serial connection

In our project we have opted not to transmit video

streams to the clients via IrDA and serial connections because of bandwidth limitations. However, compressed video data is transmitted if a wireless-LAN connection is established.

5.5 Implementation Details

The measurement of system properties can be very complex. High accuracy of the measurements requires a lot of time. The tradeoff between time and accuracy has to be taken into account.

The CPU power of a client device can be measured by counting solved mathematic equations in a given time interval (following the *linpack* idea). Measurement of communication bandwidth and response time are more complex. Response time can be measured by taking time while sending a roundtrip request of a given size. For our environment, experiments have shown that response time can also be used as an estimate for communication bandwidth in a point-to-point connection (IrDA, serial).

For our video surveillance application, we have used DCOM-RPC and RAPI connectors. DCOM-RPC is used to interconnect server- and client-side COM+/DCOM components on Windows 2000. Compressed video streams, still images and text notes are delivered to the Windows CE client via RAPI in block mode. A TCP/IP connector for communication over our wireless LAN infrastructure is currently under development.

Figure 9 shows a client of the video surveillance application for Windows CE 3.0. The client presents textual output of the motion tracker stating a person has moved (from) the room. The still image recorded at scene shift shows the empty computer lab.

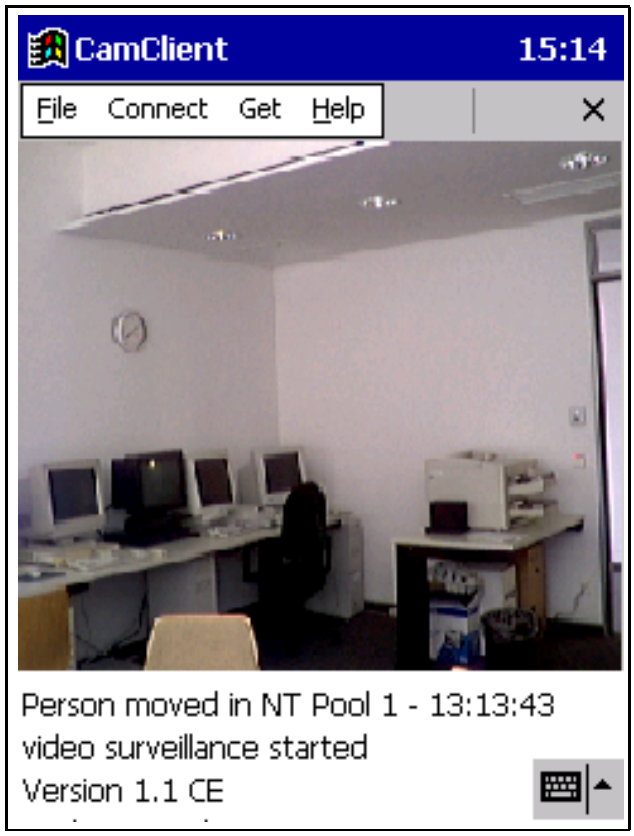

Figure 9. Windows Ce 3.0 - client

6 Related Work

While developing our configuration description languages, we have studied a variety of Module Interconnection Languages (MIL) like Darwin , PCL, Rex [7], OLan [2] Durra [1]. These Languages describe the modules and their connections. A comparison of these languages can be found in [3]. However, in contrast to our approach, these languages do not focus on mobile devices with their constantly changing communication characteristics.

Research at the University of Illinois, Urbana-Champaign [8] has been focused on a special middleware to adapt application configurations to systems properties. Applications are controlled by the middleware and re-configured on changing conditions. Again, this work is dealing mainly with resource management for standard desktop computer systems.

There exist a number of industry projects targeting the market of mobile compagnions and embedded devices. SUN Microsystems has introduced smart services with their strategy ONE. Smart services differentiate their delivered quality of service depending on a service context containing a user identity, the type of the device, and business rules.

With its .NET framework, Microsoft has introduced the ASP.NET concept which allows adaptation of the quality of a service to type of a client device. This approach provides great flexibility and supports a variety of different representations of the same information source. However, the type of device is taken as the only parameter when choosing a particular representation. This is fine for relatively static, lightweight content, such as web pages, but does not solve the problem of mobile devices with a variety of communication interfaces whose parameters may change over time (see Figure 1).

7 Conclusions

We have presented a novel approach towards configuration of component-based services for mobile systems. In contrast to existing solutions, our approach concentrates on special characteristics of mobile systems which often work in ad hoc scenarios with dynamically changing communication parameters.

We have developed an XML-based configuration language, which defines a set of rules for component configuration depending on a number of environmental parameters (constraints). The concept of connectors provides a generic interface to a variety of different interonnection media which may be proprietary and device-specific.

Our framework has been implemented on the basis of the Microsoft Distributed Component Object Model on Windows 2000 and on the Windows CE-based Pocket PC platform. As a proof-of-concept scenario we have realized a configurable distributed video surveillance application.

Additional research will focus on automatic generation of configuration profiles for distributed application as well as on extension of the framework with additional connector types.

References

[1] M. R. BARBACCI, D. L. DOUBLEDAY, C. B. WEINSTOCK, M.J. GARDNER, R.W. LICHOTA: *Durra: a structure description language for developing distributed applications*, Soft.Eng.J. 8(2) pp 83-94, March 1993.

[2] L. BELLISARD, MICHEL RIVEILL: *Olan: A Language and Runtime Support for Distributed Application Configuration*, Proc. of 16th IEEE Intl. Conf. on Distributed Computing Systems, Hong-Kong, pp. 579-585, 1996.

[3] JUDY M BISHOP *Languages for configuration programming: a comparison*, IEEE Trans. Soft Eng. to appear, also UP CS Tech Report 94/04.

[4] FANGZHE CHANG AND VIJAY KARAMCHETI: *Automatic Configuration and Runtime Adaption of Distributed Applications*, Ninth IEEE Symposium on High Performance Distributed Computing, August 2000.

[5] INTEL OPEN SOURCE COMPUTER VISION LIBRARY HOMEPAGE: *http://mrl.intel.com*

[6] GREGOR KICZALES, JOHN LAMPING, ANURAG MENDHEKAR, CHRIS MAEDA, CRISTINA VIDEIRA LOPES, JEAN-MARC LOINGTIER, JOHN IRWIN:*Aspect-Oriented Programming*, Published in proceedings of the European Conference on Object-Oriented Programming, Finland, Springer Verlag LNCS 1241, June 1997.

[7] JEFF KRAMER, JEFF MAGEE, MORRIS SLOMAN, NARANKER DULAY: *Configuring Object-Based Distributed Programs in REX*, IEE Software Engineering Journal, 7(2):139–149, March 1992.

[8] BAOCHUN LI AND KLARA NAHRSTEDT: *QualProbes: Middleware QoS Profiling Services for Configuring Adaptive Applications*, in Joseph S. Sventek, Geoff Coulson (Eds.): Proceedings of Middleware 2000, IFIP/ACM International Conference on Distributed Systems Platforms, New York, NY, USA, April 4-7, 2000, LNCS 1795 Springer 2000.

[9] JEFF MAGEE, NARANKER DULAY, SUSAN EISENBACH, JEFF KRAMER: *Specifing Distributed Software Architektures* , In Proceedings of 5th European Software Engineering Conference (ESEC 95). 1995. Sitges, Spain.

[10] MARTIN MEYKA: *Replikation als Aspekt in verteilten Systemen*, Diploma Thesis, Dept. of CS, Humboldt University Berlin, July 2000.

[11] MATTHIAS RADESTOCK, SUSAN EISENBACH: *Agent-based Configurating Management*, In Proc. of the 7th IFIP/IEEE Int. Workshop on Distributed Systems: Operation and Management, 1996.

[12] MICROSOFT WINDOWS EMBEDDED HOMEPAGE: *http://www.microsoft.com/ windows/embedded*

[13] WORLD WIDE WEB CONSORTIUM HOMEPAGE: *www.w3.org*