Automated Context-Driven Composition of Pervasive Services to Alleviate Non-Functional Concerns

Davy Preuveneers and Yolande Berbers
Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Leuven, BELGIUM
[davy.preuveneers, yolande.berbers]@ cs.kuleuven.be

Abstract: *Service-oriented computing is a new emerging computing paradigm that changes the way applications are designed, implemented and consumed in a ubiquitous computing environment. In such environments computing is pushed away from the traditional desktop to small embedded and networked computing devices around us. However, developing mobile and pervasive services for a broad range of systems with different capabilities and limitations while ensuring its users a minimum quality of service is a daunting task. The core contribution of this paper is a context-driven composition infrastructure to create an instantiation of a pervasive service customized to the preferences of the user and to the capabilities of his device. We implement services as a composition of components. This enables us to compose a service implementation targeted at a specific device while still being able to adapt it at run-time to respond to changing working conditions.*

Keywords: *Ubiquitous Computing, Context, Composition, Adaptation, Component-based Services.*

Received: July 30, 2005 | ***Revised:*** August 10, 2005 | ***Accepted:*** August 25, 2005

1. Introduction

The third wave of computing is slowly appearing: ubiquitous computing. This new computing paradigm promises an augmented reality that changes the way people interact with computers. It promises continuous human computer interaction with small embedded and networked computing systems around us that provide 24/7 access to information and computational capabilities, and envisions pushing computing away from the traditional desktop system [28].

When Weiser introduced the area of ubiquitous computing [29] in 1991, he put forth a vision of a deployment of devices at varying scales, ranging from handheld personal digital assistants to larger shared devices providing the necessary infrastructure support. He also envisioned a new paradigm of interaction using natural interfaces such as speech, video and sensor inputs, instead of the traditional keyboard and mouse, to facilitate communication between humans and computers.

This new way of interaction with computational devices introduces two important non-functional requirements with respect to the development of pervasive services. The first requirement is that services should not be developed from scratch to provide an optimal implementation for each device with different capabilities and limitations. As such, the design and the deployment process of services should offer the software engineer the necessary flexibility to target services to a broad range of systems, ranging from personal handhelds and smart phones to larger set-top boxes. The second requirement is that applications and services need to be less dependent on user-intervention and prevent users from being overwhelmed with intrusive human computer interactions. Therefore, service-oriented architectures must be able to gather information about the user and his physical and digital environment to autonomously adapt the behavior of the provided services according to the current context of the user and the device he is interacting with. It are these two gaps that our infrastructure is trying to fill in.

To alleviate the first concern we employ a component-based software engineering methodology [21] for building services to be consumed within ubiquitous computing environments. Component-based software engineering is being recognized as an important approach for software upgrade and dynamic reconfiguration in dependable resourceconstrained and embedded devices. As such, services are a collection of inter-connected computational building blocks that offer a particular functionality to a network of devices with varying capabilities. The second concern with respect to customization of services is tackled by context-driven composition of components and using context-awareness, including user preferences, to increase the non-intrusiveness of pervasive services. The overall objective is to achieve automatic composition of customized pervasive services using components as building blocks by using contextual information [16] for dealing with the variability of pervasive computing devices and user personalization.

In section (2) we discuss several non-functional concerns with respect to services being targeted at ubiquitous computing environments and how a component-based development methodology can be of help for designing and deploying pervasive services. Section (3) discusses how context, including user preferences and resource availability among other information, and components are formally specified in OWL to support automated service composition for optimal deployment on a specific device. The core ideas of automated context-driven composition of components into services are directly illustrated in section (4) and build upon previous work on context modeling [16], pervasive service specification [14] and context management support [15]. In section (5) we evaluate our composition infrastructure. Section (6) provides an overview of related work. We end with conclusions and future work in section (7).

2. Non-functional concerns of pervasive services

Pervasive services offer a certain functionality to nearby customers and are accessed in an *anytime-anywhere* fashion, while being deployed on all kinds of devices. The aspects of user mobility, personalization and context-awareness may activate service adaptation and migration to other devices with different characteristics. In this section we review several non-functional concerns with respect to ensuring that all the requirements of the delivery and provisioning of the service to be consumed in a mobile and pervasive setting are met.

2.1 Resource-awareness

In the ubiquitous computing setting, available services will approach the user on detection of his presence, while the user wants to have the best deals. However, this multi-user and multi-computer interaction causes a competition of resources on shared devices. Therefore, resource-awareness about the maximum availability and current usage of processing power, memory, network bandwidth, battery lifetime, etc., is a prerequisite to guarantee a minimum quality of service. Due to the black box nature of components, a component's functional properties (interfaces and task description) as well as its non-functional properties (resource requirements and adaptation policies) [30] can be specified more easily to better support resource-awareness.

2.2 Mobility

User mobility is a corner stone of the society of tomorrow. Due to possible wireless network disruptions, a user may wish to download and run a service locally. If, on the other hand, the downloaded service does not run within the currently available resources of the device, the user may wish to run the service on a remote more powerful device in the vicinity of the user or to relocate (parts of) an already running service. In both cases the mobile setting of the user triggers service migration to other devices.

The encapsulation of the implementation of components and their message-based interaction make it easy to relocate a component for distributed execution of the service by instantiating the component elsewhere [17] and rerouting the messages.

2.3 Adaptation

In the face of highly dynamic environments, heterogeneous devices and their changing context, services will need to adapt to changing working conditions.

For stateless components, a component can be replaced with a similar component at runtime as long as the syntax and semantics of the interfaces of a component remain the same [27]. Other components may require state transfer first.

2.4 A simple component-based application

Our pervasive service design methodology makes use of the SEESCOA component methodology [25] which is targeted at software development for embedded systems. This implies that a pervasive service incorporates components and connectors to fulfill its functional aspects. *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* [30] 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.

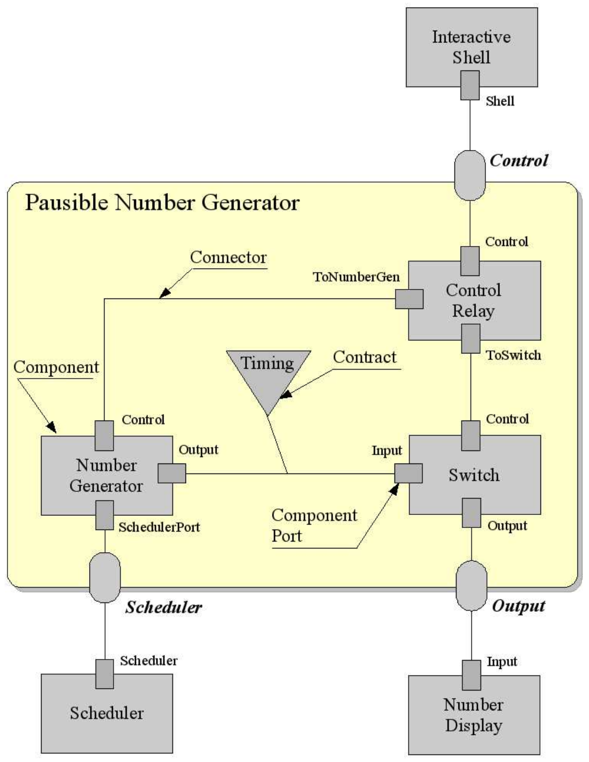

Figure 1. A simple component-based application.

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 shown on a screen by the *Number Display* component. 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.

3. Integrating context-awareness into component-based pervasive services

The example in Figure 1 is an extension of an even smaller application that does not include the switch and control relay. In the latter case, the user is not able to pause the random number generation.

The core of our contribution is that pervasive services are modeled by specifying all the components, including the optional components, and that these components are instantiated by selecting an appropriate implementation, or bypassed depending on the current context of the device or any user requirements. The functional specification of components and runtime support for their non-functional concerns in component systems allow the programmer to design services in terms of components and focus on program logic instead of deployment dependencies. With proper middleware support, the programmer does not need to implement resource monitoring, decision making or adaptation of services to respond to changing working conditions.

Two prerequisites for context-aware composition of pervasive services are an explicit model for specifying context and middleware that is able to gather and interpret this contextual information. In the Context Toolkit [4], context is modeled as a set of key-value pairs. The more structured approaches for modeling context that have been proposed in the past use RDF [11], UAProf and CC/PP [10], and CSCP [2]. Ontologies, which allow the definition of more complex context models, have been used in several context modeling approaches [3,7,19].

We have designed a context ontology [16] in OWL based on the concepts of *User*, *Platform*, *Service* and *Environment*. This ontology is specifically targeted at context-driven adaptation of mobile services [23]. This context ontology is used in our context management system which is discussed in [15]. The context management system, which in itself is also component-based, provides all the necessary information for context-based composition and adaptation of pervasive services. Pervasive services in our methodology are more than just a collection of components with specific responsibilities. In [14] we provide a more detailed specification of pervasive services. In short, pervasive services act as normal composite components but with the following extra dedicated ports:

Service Information Interface: the Service Information Interface provides a static description of the semantics and syntax of a service and its service ports, and hence of how the service can be interfaced, so that other components or services can discover and use the service. This information is expressed in OWL-S [24]. This port is used to gather information about resource requirements and adaptation policies.

Service Control Interface: The Service Control Interface is a standard dedicated interface for controlling a service. It allows the service to be (re)started, updated, relocated, stopped and uninstalled. By making this an obligatory interface, no knowledge about the other service ports is required for basic service management. This port is used to alleviate the adaptation concern.

Context Interface: The Context Interface is responsible for the sending and receiving of the context information available at run-time when the service is active. Among other things, it allows the service to be notified of new resources, and to inform other services or devices about resources currently in use by this service.

By modeling services as components, all aspects of composing components into a service apply to services is well, which means that services can be combined to form a new service.

```
<owl:Class rdf:ID="SimpleComponent">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="&xsd;#int">
      1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasPort" />
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="CompositeComponent">
  <rdfs:subClassOf rdf:resource="#SimpleComponent" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="&xsd;#int">
        1
      </owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasComponentInstance" />
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:about="#hasComponentInstance">
  <rdfs:domain rdf:resource="#CompositeComponent" />
  <rdfs:range rdf:resource="#ComponentInstance" />
</owl:ObjectProperty>

<owl:Class rdf:ID="Port">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="&xsd;#int">
        1
      </owl:cardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="maxInstances" />
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="PortGroup">
  <rdfs:subClassOf rdf:resource="#Port" />
</owl:Class>
```

Figure 2. Excerpt of the meta-model specification of a component in OWL.

The automated composition of our infrastructure requires that all components are fully specified. We therefore created a component ontology in OWL which serves as a meta-model of all the concepts of the SEESCOA methodology, specifying components, ports, messages, parameters, connectors, contracts, etc. as an *OWL Class*, *ObjectProperty* or *DatatypeProperty* with cardinality restrictions where appropriate. See Figure 2 for a partial meta-model specification of a component in OWL[1]. It allows the software engineer to validate component descriptions by using a regular OWL validator [1] so that for automated composition the infrastructure can rely on correctly specified components. As components are self-contained and perhaps independently developed, we have to make sure that messages sent out by one component port are well understood by the other component port on the receiving part of the connector. We therefore also declare all message types as concepts in another OWL file. This concept ontology is necessary as messages are not restricted to primitive data types, such as strings and integers, but can also include objects of any kind of which the inheritance properties can be modeled in the concept ontology as well. The component and concept ontology allow us to make sure that alternative implementations of the same blueprint match syntactically and semantically and that connected component ports exchange messages that are understood by both parties.

```
<profile:serviceParameter>
  <compprofile:RequiredResources rdf:ID="requiredMemory">
    <profile:sParameter>
      <context:Memory>
        <context:bytes>1048576</context:bytes>
      </context:Memory>
    </profile:sParameter>
  </compprofile:RequiredResources>
</profile:serviceParameter>
```

Figure 3. Minimum memory requirement as ServiceParameter in OWL-s.

Context-awareness comes into play during the evaluation of the non-functional properties of services and components. The service and each component separately can specify minimum resource requirements, such as the minimum availability of memory, processing power or network bandwidth. These resource requirements are specified by extending the *serviceParameter* concept in OWL-S of which an example is given in bold in Figure 3. In this example, a minimum memory requirement for the service is specified, but it could have been any contextual requirement with respect to concepts in our context ontology [16]. For example, user preferences in a certain context with respect to a service could be taken into account to increase the non-intrusiveness of the service.

Our context-driven composition infrastructure instantiates a pervasive service by connecting a selection of components such that all requirements hold. In the next section, we will describe an example of a service with several optional components that can be activated when possible. These optional components require extra resources, and we therefore specify triggers to model QoS requirements and to define adaptation policies.

[1] The OWL meta-model specification of a component can be found at http://www.cs.kuleuven.be/~davy/owl/Component.owl

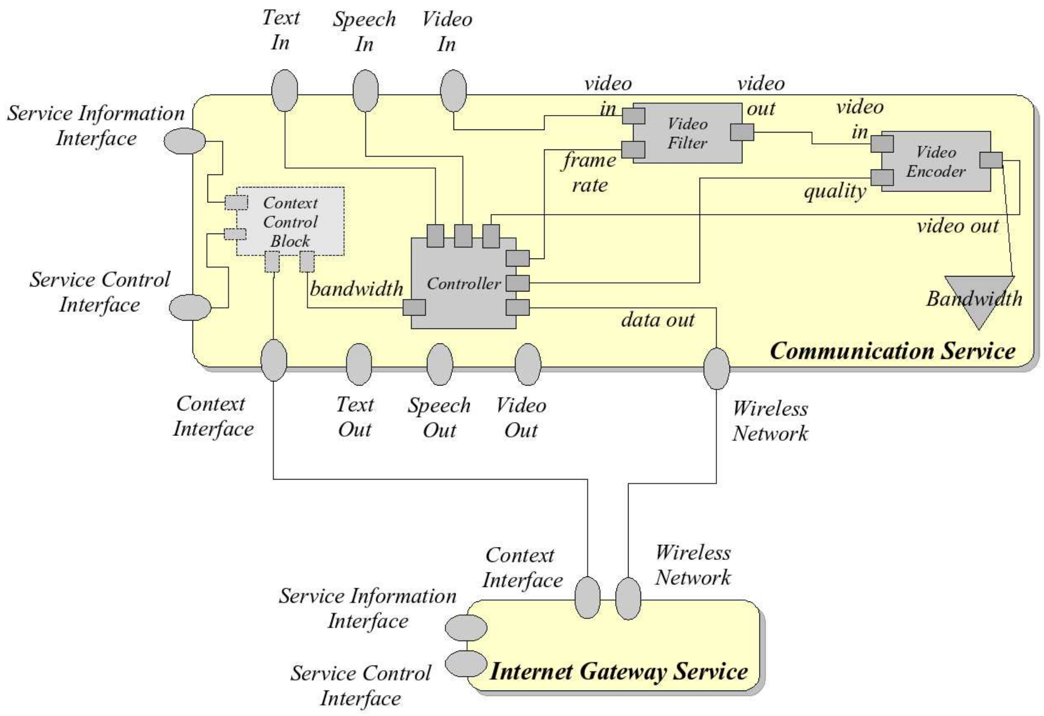

Figure 4. Internet Gateway Service and Communication Service.

4. Context-driven composition of pervasive services

The core ideas of how our infrastructure supports automated context-driven composition of pervasive services are discussed in this section. The concepts of taking into account the capabilities and limitations of a device, resource awareness and user preferences in a specific context are illustrated by means of an example. We also provide an outline of our algorithm to discuss the steps taken to deploy a customized pervasive service and mention implementation details.

Consider the following scenario in a ubiquitous and mobile computing environment where two friends, Jack and Jill, are having a conversation over the Internet:

"Jack is heading off to work and using public transport. He is using his advanced smart phone and a shared Internet gateway service on the train. Bandwidth is equally shared by all passengers currently logged on to the Internet gateway service. Jill is at home using her laptop with broadband Internet connection and has no resource limitations.
*The communication service both Jack and Jill downloaded on their device is able to communicate using text messaging, speech and video, sorted by increasing bandwidth requirements. The speech and video quality (*high, medium *and* low*) depends on the compression, the frame rate and the frame size being used. These parameters are chosen dynamically depending on the available bandwidth and processing power on the smart phone or are defined as user preferences in a certain context."*

4.1 The component-based model of the communication service

See Figure 4 for a simplified overview of both services. We will now focus on the *Communication Service* for which a composition will be auto-instantiated depending on the current context of the user and the communication device. The service has the following components:

Audio Encoder and Decoder: Adaptable and optional components for (de-)compressing the audio stream with high, medium or low quality encoding.

Video Filter: Optional components for reducing the video frame rate or changing the frame size. See Figure 5 for an instantiation of the OWL component meta-model.

Video Encoder and Decoder: Adaptable and optional components for (de-)compressing the video stream with high, medium or low quality encoding.

Controller: (de-)multiplexes text, speech and video, and sends/receives the combined data stream.

```
<component:SimpleComponent rdf:ID="FrameRateFilter">
  <component:hasPort rdf:resource="#FrameRate" />
  <component:hasPort rdf:resource="#VideoIn" />
  <component:hasPort rdf:resource="#VideoOut" />

  <component:PortGroup rdf:ID="FrameRate">
    <component:maxInstances rdf:datatype="&xsd;#int">
      1
    </component:maxInstances>
    <component:hasMessage>
      <component:InMessage rdf:ID="SetFrameRate">
        <component:hasParameter>
          <component:Parameter rdf:ID="FramePerSec">
            <component:parameterType rdf:datatype=
              "&xsd;#anyURI">&xsd;#int
            </component:parameterType>
          </component:Parameter>
        </component:hasParameter>
      </component:InMessage>
    </component:hasMessage>
  </component:PortGroup>

  <component:PortGroup rdf:ID="VideoIn">
    <component:maxInstances rdf:datatype="&xsd;#int">
      1
    </component:maxInstances>
    <component:hasMessage>
      <component:InMessage rdf:ID="NewFrame">
        <component:hasParameter>
          <component:Parameter rdf:ID="Frame">
            <component:parameterType rdf:datatype=
              "&xsd;#anyURI">&concepts;#VideoFrame
            </component:parameterType>
          </component:Parameter>
        </component:hasParameter>
      </component:InMessage>
    </component:hasMessage>
  </component:PortGroup>

  <component:MulticastPort rdf:ID="VideoOut" />
    <!-- Similar to the video-in port -->
  ...
</component:SimpleComponent>
```

Figure 5. Instantiation of the OWL component meta-model for a video frame rate filter component.

The audio and video components in this service model are optional, and can be activated when enough resources are available. These components can have several implementations using different encoding schemes. Both parties would have to agree which encoding schemes can be used on the two communication devices. However, in this example one party, Jill, has no resource restrictions and can instantiate any component she likes.

4.2 Resource requirements for deployment

The minimum resource requirements for the service, as shown in bold in Figure 3, specify what is needed to have a text-based conversation. This kind of communication requires the least bandwidth and processing power. If more resources are available, audio and video-based communication can be enabled as well. Information about available resources is provided by our context management system [15]. In this example, the limiting factors are the processing capabilities of the smart phone and the available bandwidth for each passenger. An overview of all resource requirements is given in Table 1. The total bandwidth requirements for video streaming depend on the quality options and the use of other video filters. The processing power requirements are only an estimate as a real device may have hardware support for multimedia applications. Our prototype is implemented in the Java language and makes use of other pure Java libraries. Therefore, the system requirements are a lot higher to process audio and video on demand in real-time.

4.3 User preferences in a certain context

A user may have a preference stating that during office hours the high quality option for video encoding should be used. See Figure 6, the preference property and value as well as the time and location conditions are marked in bold. Using this contextual information, an appropriate service is instantiated that takes any user preferences and the current available resources and the limitations and capabilities of the device into account. For example, before deploying the communication service on a PDA or smart phone, we check if the hardware has the necessary support for video communication as, for example, not all devices have a camera on-board.

```
<context:UserPreference rdf:ID="VideoEncoding1">
  <context:prefProperty rdf:resource="#VideoQuality" />
  <context:prefValue rdf:resource="&concepts;#High" />
  <context:prefCondition>
    <context:Location>
      <context:where rdf:resource="&concepts;#Office" />
    </context:Location>
  </context:prefCondition>
  <context:prefCondition>
    <context:Time>
      <context:when rdf:resource="&concepts;#NineToFive" />
    </context:Time>
  </context:prefCondition>
</context:UserPreference>
```

Figure 6. User preference with respect to video encoding.

4.4 Outline of the context-driven service composition procedure

The goal of our context-driven composition infrastructure is an automated instantiation of a service targeted at the capabilities of a specific device that takes into account any user preferences in a specific context. The whole algorithm is entirely based on the processing of OWL and OWL-S specifications and consists of the following steps.

(1) Hardware: Process the hardware description of the device to discover resource specifications, such as maximum available memory, processing power, network bandwidth, etc., as well as support for user input and output, such as microphone, camera, speaker, keyboard, display, sensors, etc. This is part of the context specification of a device and is largely static information.

(2) Resource awareness: Request the current available resources from the context management system. This is also part of the current context, but mostly dynamic information achieved by monitoring the system.

Table 1. Non-functional requirements and restrictions of the communication service.

Component	Optional	User Preference	Processing Power	Bandwidth
Text-based Messenger	False	-	≈ 50 MIPS	> 0 bps
Audio Encoder/Decoder	True	High Quality Medium Quality Low Quality	≈ 200 MIPS	64 kbps 32 kbps 8 kbps
Video Encoder/Decoder	True	High Quality Medium Quality Low Quality	≈ 800 MIPS	10:1 Reduction 20:1 Reduction 30:1 Reduction
Frame Resizer	True	-	≈ 100 MIPS	4:1 Reduction
Frame Rate Filter	True	-	≈ 50 MIPS	2:1 Reduction

(3) Service requirements: Check any I/O requirements of the service with the I/O support of the device. Check if the minimal resource requirements can be fulfilled. If deploying is not possible, then stop or propose to run the service on another device in the neighbourhood or relocate already running services.

(4) User preferences: Check for all preferences if the context of application is fulfilled and if these preferences can be enforced given the available resources. Checking if the context applies, may require some reasoning steps, for example, to determine that geographical coordinates of a GPS system point to a location known as our office.

(5) Component selection: Given the available resources, eliminate all optional components which resource requirements are too demanding. For each obligatory component, retrieve the resource requirements for all of its implementations.

(6) Constraint solving: Given the above resource constraints of the device and the requirements for the obligatory components and user preferences, solve the set of equations to find a minimal composition of component instances. Cut down on the user preferences if no solution is found.

(7) Adaptation policy: Check which optional components can be enabled for the preliminary minimal service instantiation from the previous step. Given the available resources, these optional components can be enabled immediately, when enough resources are available or when the context changes and other user preferences are to be applied.

For the multimedia components of the communication service, we have made use of the IBM Mpeg4 Java toolkit [9] and for the OWL parsing and reasoning the Jena 2 framework [8] is used. The necessary infrastructure support for instantiating a customized service based on the service model was developed on top of Draco [26], an extensible runtime system for components designed to be run on embedded devices. It acts as a component container that instantiates components and manages all connections between components. It has runtime support for non-functional concerns, such as component distribution, live updates, contract monitoring and resource management. This runtime environment with extensions provides a unique test platform for validating the proposed service concepts in a ubiquitous computing context.

5. Evaluation

The case study has shown that for varying hardware and resource descriptions the implemented infrastructure is able to compose and instantiate a service while also taking the user preferences into account. Another advantage is that composing the service targeted at a specific device can also be carried out on a more powerful device when composing the service, for example, on a personal handheld would take too long.

Something that is currently not implemented is multi-party negotiation of which components to instantiate as there can be dependencies between component deployments on different devices. In the above example, there is no use in instantiating a video encoding component on one device if the other party is not able to process or show the video stream.

The two non-functional concerns with respect to resource-awareness and user preferences in a certain context are easily managed using a component-based methodology if all aspects are fully specified. These aspects are specified by hand in OWL using the Protégé [18] tool. We acknowledge that this is not a user friendly way to declare user preferences.

However, not all non-functional aspects can be solved that easily using a component-based software engineering methodology. For cross-cutting concerns, such as security or logging, an aspect-oriented software development approach (AOSD) can prove to be more useful. Security concerns, for example, relate to the connection between components and may impose restrictions on how to customize and adapt pervasive services, such as restricting the relocation of parts of a service to another device to avoid interception of sensitive messages sent over a network. The disadvantage of using AOSD techniques is that depending on the tools applied, such as AspectJ [22], the technology can be quite invasive while modifying the component in such a way that other non-functional concerns, such as resource requirements, are no longer guaranteed to be correct.

6. Related work

Two very well-known component models are OMG's CORBA Component Model (CCM) [13] and Sun's Enterprise JavaBeans (EJB) [20]. However, both only provide limited support for non-functional concerns, such as persistency, transactions and access control. Neither of them provides support for dynamic selection of component instantiations.

The COMQUAD component model [6] provides support for non-functional aspects and describes them orthogonally to the application structure using descriptors. These non-functional aspects are woven into the application using an AOSD approach based on the non-functional properties of components. Additionally, the COMQUAD component model has extensions to provide support for stream-based communication. However, selecting components based on functional properties is not the goal of the authors. In our composition infrastructure, both functional and non-functional properties need to be considered to guarantee interoperability. The authors are working on providing support for adaptable behaviour, as already supported in our infrastructure using composite components. The authors also acknowledge the interference between functional and non-functional aspects and between different non-functional concerns. Additionally, our infrastructure supports automated component selection and composition using a context-driven approach.

Efstratiou et al. [5] propose an architecture with support for context-aware service adaptation. The authors describe the architectural requirements for adaptation control and coordination for mobile applications. The adaptation mechanisms to respond to, for example a change in resource availability, are less flexible than ours as the authors define operational modes of an application together with a contextual trigger that would make the application switch to that mode. This coarse-grained adaptation mechanism allows them to use very simple XML descriptions of a service and operation mode. Our approach does not require the software engineer to define all possible instantiations of a service. Our composition infrastructure takes care of finding an appropriate composition, which additionally supports taking user preferences into account. Our infrastructure allows operation modes that the software developer originally may have not yet foreseen.

Lin et al. [12] propose a goal description language for automatic composition of semantic web services. The authors acknowledge that it is hard to adapt to users' requirements in current web service compositions, and propose a goal description language based on the OWL ontology description language to describe the goals that need to be achieved, relationships within goals, constraints for achieving the goals, and a way to detect any inconsistencies in the specified goals. However, this language for automatic composition is targeted at web services and does not consider the non-functional requirements of pervasive services as mentioned in a previous section.

7. Conclusion and future work

We have developed infrastructure support for automated context-driven composition of components into pervasive services, while taking into account non-functional concerns such as the capabilities and limitations of a device on which it will be deployed, resource availability and user preferences in a specific context. It lets the software engineer focus on the application logic of smaller building blocks, while optimal deployment of a service on a specific device and adaptation support for responding to changing working conditions is managed by our supporting infrastructure.

First, we have identified several non-functional concerns with respect to pervasive services and then discussed how a component-based methodology can alleviate these concerns. We formalized the component methodology by creating a meta-model of all software aspects of components to allow automated processing of the functional and non-functional properties of components and services. We discussed how context-awareness can be used for composition of component-based services using our previous work on context modelling [16], specification of pervasive services [14] and context management support [15]. We illustrated the core ideas of our infrastructure support by means of an example, and gave an outline of all the steps in the procedure to achieve a customized service. We evaluated our work and can conclude that for targeting a specific platform and incorporating user preferences, our automated context-driven composition infrastructure is able to provide the necessary support for these non-functional concerns typical for pervasive services.

However, more work should be carried out to provide support for cross-cutting non-functional concerns, such as security. Security requirements cannot be derived from component descriptions as is the case for resource requirements. Future work in the short term will focus on how to integrate multi-party negotiation for selecting and instantiating components in the presence of dependencies between component deployments on different devices.

References

[1] BBN Technologies, OWL Validator version 20040716, July 2004.

[2] S. Buchholz, T. Hamann, and G. Hubsch, Comprehensive Structured Context Profiles (CSCP): Design and Experiences, in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, March 2004.

[3] H. Chen, T. Finin, and A. Joshi, An Ontology For Context-Aware Pervasive Computing Environments, in *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 2003.

[4] A. K. Dey, D. Salber, and G. D. Abowd, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, *Human-Computer Interaction (HCI) Journal*, vol. 16, no. 2-4, pp. 97-166, 2001.

[5] C. Efstratiou, K. Cheverst, N. Davies, and A. Friday, An architecture for the effective support of adaptive context-aware applications, in *Proceedings of 2nd International Conference in Mobile Data Management*, vol. Lecture Notes in Computer Science Volume 1987. Hong Kong: Springer, January 2001, pp. 15-26.

[6] S. Göbel, C. Pohl, S. Röttger, and S. Zschaler, The COMQUAD component model: enabling dynamic selection of implementations by weaving non-functional aspects, in *AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development*. New York, NY, USA: ACM Press, 2004, pp. 74-82.

[7] T. Gu, et al., An ontology-based context model in intelligent environments, In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, San Diego, California, USA, January 2004.

[8] HP Labs, Jena 2 - A Semantic Web Framework, 2004.

[9] IBM alphaWorks, IBM Toolkit for MPEG-4, *2005.*

[10] J. Indulska, et al., Experiences in using CC/PP in context-aware systems, in *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science (LNCS), J.-B.Stefani, I. Dameure, and D. Hagimont, Eds., vol. 2893. Springer-Verlag, November 2003, pp. 224-235.

[11] P. Korpipää, et al., Managing context information in mobile devices, *IEEE Pervasive Computing, Mobile and Ubiquitous Systems*, vol. 2, no. 3, pp. 42-51, July-September 2003.

[12] M. Lin, H. Guo, and J. Yin, Goal description language for semantic web service automatic composition, in *SAINT*, 2005, pp. 190-196.

[13] Object Management Group, CORBA Component Model, v3.0, 2005.

[14] D. Preuveneers and Y. Berbers, Semantic and syntactic modeling of component-based services for context-aware pervasive systems using OWL-S, in *Proceedings of the 1st International Workshop on Managing Context Information in Mobile and Pervasive Environments*, Cyprus, May 2005, pp. 30-39.

[15] D. Preuveneers and Y. Berbers, Adaptive context management using a component-based approach, in *LNCS 3543: Proceedings of 5th IFIP International Conference on Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science (LNCS), L. Merakos, N. Alonistioti, and L. Kutvonen, Eds., vol. 3543. Springer-Verlag, June 2005, pp. 14-26.

[16] D. Preuveneers, J. Van den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. De Bosschere, Towards an extensible context ontology for Ambient Intelligence, in *LNCS 3295: Proceedings of the Second European Symposium on Ambient Intelligence*, ser. Lecture Notes in Computer Science (LNCS), P. Markopoulos, B. Eggen, E. Aarts and J.L. Crowley, Eds.. vol. 3295. Springer-Verlag, November 2004, pp. 148-159.

[17] P. Rigole, Y. Berbers, and T. Holvoet, Mobile adaptive tasks guided by resource contracts, in *the 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing*, Canada, October 2004, pp. 117-120.

[18] Stanford Medical Informatics, The Protégé Ontology Editor and Knowledge Acquisition System, 2005.

[19] T. Strang, et al., CoOL: A Context Ontology Language to enable Contextual Interoperability, in *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science (LNCS), J.-B. Stefani, I. Dameure, and D. Hagimont, Eds., vol. 2893. Springer-Verlag, November 2003, pp. 236-247.

[20] Sun Microsystems, The Enterprise JavaBeans 2.1 specification, 2005.

[21] C. Szyperski, *Component Software: Beyond Object-Oriented Programming, 2nd edition*. Addison-Wesley and ACM Press, 2002.

[22] The AspectJ project, AspectJ - Crosscutting objects for better modularity, 2005.

[23] The DistriNet, EDM, ELIS-PARIS, PROG and SSEL research groups, CoDAMoS: Context-Driven Adaptation of Mobile Services, *http://www.cs.kuleuven.ac.be/~distrinet/projects/CoDAMoS*, October 2003.

[24] The OWL Services Coalition, OWL-S: Semantic Markup for Web Services, Release 1.1, November 2004.

[25] 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*, 2001, pp. 685-690.

[26] Y. Vandewoude, Draco: A middleware platform for component-oriented applications, http://www.cs.kuleuven.ac.be/~yvesv/?q=Draco, 2005

[27] Y. Vandewoude and Y. Berbers, Run-time evolution for embedded component-oriented systems, in *Proceedings of the International Conference on Software Maintenance*, B. Werner, Ed. Canada: IEEE Computer Society, October 2002, pp. 242-245.

[28] M. Weiser, The world is not a desktop, *Interactions*, pp. 7-8, January 1994.

[29] M. Weiser, The Computer for the Twenty-First Century, *Scientific American*, pp. 99-104, September 1991.

[30] A. Wils, J. Gorinsek, S. Van Baelen, Y. Berbers, and K. DeVlaminck, Flexible Component Contracts for Local Resource Awareness, in *ECOOP 2003 Workshop on resource aware computing*, C. Bryce and G. Czajkowski, Eds., July 2003.

Davy Preuveneers received his M.S. degree in Computer Science in 2002, and his M.S. in Artificial Intelligence in 2003 from the Katholieke Universiteit Leuven in Belgium. Since 2003, he is working towards his PhD at the Department of Computer Science of the Katholieke Universiteit Leuven. His research is focused on context-aware service interaction and adaptation, in particular in mobile and pervasive computing environments.

Prof. Dr. Ir. Yolande Berbers received her PhD degree from the Katholieke Universiteit Leuven in 1987. She is currently a full professor at the Department of Computer Science at the Katholieke Universiteit Leuven and a member of the DistriNet research group. Her research interests include software engineering for embedded and real-time systems, model-driven architecture, context-aware computing, distributed and parallel systems and distributed computing, and multi-agent systems with emergent behaviour. She runs several projects in cooperation with other universities and/or industry. She teaches several courses on programming real-time and embedded systems, and on computer architecture.