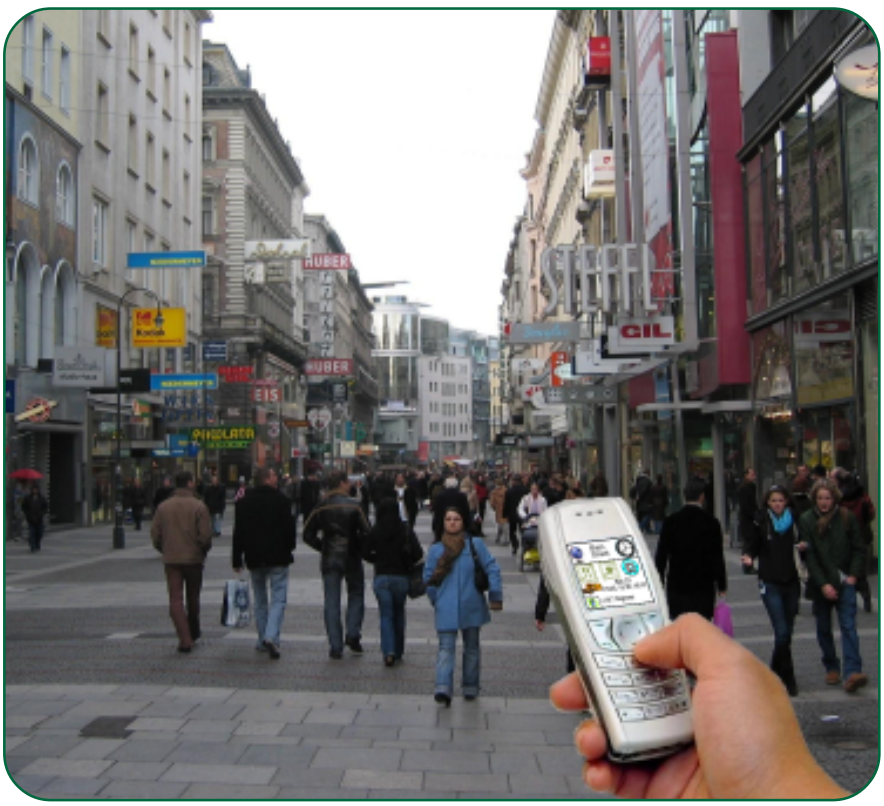
ESPOO 2007 VTT PUBLICATIONS 658



Marko Palviainen

Technique for dynamic composition of content and context-sensitive mobile applications

Adaptive mobile browsers as a case study

VTT PUBLICATIONS 658

Technique for dynamic composition of content and context-sensitive mobile applications

Adaptive mobile browsers as a case study

Marko Palviainen

Thesis for the degree of Doctor of Technology
to be presented with due permission for public examination
and criticism in Tietotalo Building, Auditorium TB109
at Tampere University of Technology,
on the 23th of November 2007 at 12 o´clock noon

ISBN 978-951-38-7051-5 (soft back ed.)
ISSN 1235-0621 (soft back ed.)

ISBN 978-951-38-7052-2 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 3, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 3, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 3, P.O.Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax + 358 20 722 4374

VTT, Tekniikankatu 1, PL 1300, 33101 TAMPERE
puh. vaihde 020 722 111, faksi 020 722 3365

VTT, Teknikvägen 1, PB 1300, 33101 TAMMERFORS
tel. växel 020 722 111, fax 020 722 3365

VTT Technical Research Centre of Finland, Tekniikankatu 1, P.O. Box 1300, FI-33101 TAMPERE, Finland
phone internat. +358 20 722 111, fax +358 20 722 3365

Technical editing Leena Ukskoski

Edita Prima Oy, Helsinki 2007

Palviainen. Marko. Technique for dynamic composition of content and context-sensitive mobile applications. Adaptive mobile browsers as a case study. Espoo 2007. VTT Publications 658. 233 p.

Keywords dynamic composition, task-based composition, mobile application development, adaptive application, adaptive browser, content and context-sensitive application

Abstract

The mobile environment brings new challenges for applications. Mobile usage is spontaneous and applications should be fast to install, start, and use in mobile devices and wireless networks. The wireless network connections offer typically less bandwidth than fixed line connections and may cause costs for the user. In addition, the input and output capabilities and memory and processing power resources of mobile devices are typically limited in comparison to desktop computers. This all sets requirements for adaptation methods that could provide more usable and efficient applications for specific users, contexts, and services available on the Web.

Implementation of adaptive applications requires methods for context-sensing and adaptation. The context can change rapidly when a user is moving in a physical environment. Hence, methods that can fast adapt an application for a rapidly changing context are needed. An adaptive application should learn about user behaviour, sense the activity of the user, and use the idle time of the application for *speculative adaptation* that prepares application parts for potential future contexts in the background. In addition, errors may arise while an adaptive application is being composed for a new context. For example, if a mobile device is disconnected, it is not possible to fetch contents from the Web.

In this dissertation it is argued that the task-based composition technique helps developers to construct adaptive applications for mobile devices and makes the dynamic composition of content and context-sensitive applications more fluent. The task-based composition technique is based on the model of the World Wide Web Consortium (W3C) that provides a *requestor-adaptor* structure for content adaptation. Like the requestor-adaptor element of the W3C model, a *task* is an adaptation element that can provide additional context information, request other tasks, adapt their responses, and deliver new or refreshed responses for the

requestors. Tasks can prepare content and context-sensitive application instances for current and predicted contexts in many phases and finally compose an application of the prepared instances. The task-based composition technique extends the W3C model with *task factories* that construct tasks for adaptation requests and specific contexts. Tasks are defined with an XML-based language that enables developers to describe tasks and context-sensitive adaptation actions and their settings. Both context-sensitive tasks and application instances can be cached, which can speed up the adaptation of applications. This dissertation focuses on adaptive browsers that are constructed for mobile devices and discusses how the task-based composition technique can support client-side dynamic composition of content and context-sensitive applications and improve performance when UIs are adapted for rapidly changing contexts and for services available on the Web.

Preface

The work reported in this dissertation was carried out in the SW tools team at VTT Technical Research Centre of Finland during 2001–2007. One of the initial aims of this work has been to support other mobile Internet research projects at VTT by providing a research and development environment for new mobile applications. The work described in this thesis has been done within the contexts of “ALLLAS”, "WAPproxy", "WAP Multimedia", “AUTOSPACE”, and “COSI” projects. I am grateful for the financial support provided by VTT and Tekes and for various Finnish industrial companies that participated in the projects. Additional support was received from the Ulla Tuominen Foundation, which awarded a scholarship for the work in 2005. Many individuals contributed to this research.

At first, I want to warmly thank, Professor Kai Koskimies from Tampere University of Technology for supervising my work. Thank you for guidance, valuable comments, and encouragement during my work. Many thanks also to Professors Tarja Systä, Tommi Mikkonen, and Ilkka Haikala from Tampere University of Technology for inspiring discussions and comments and suggestions on this work. In addition, I want to thank Professor Keijo Ruohonen for help in some calculations that are presented in this thesis.

I was honoured to have Professor Jari Porras from the Lappeenranta University of Technology and Professor Ivica Crnkovic from Mälardalen University as the pre-examiners of my dissertation. I am grateful for their expert comments which led to remarkable improvements in the thesis. Their insightful and constructive comments helped in clarifying the focus of this thesis and in recognising deficiencies in the argumentation. Also, I want to thank Timo Laakko, who was the project manager in the projects studying mobile Internet. I wish to thank him for the invaluable advice, assistance, and support that helped me to carry out this thesis.

This work has benefited from the help and advice of several people from VTT and people attended the above mentioned projects. I would like to thank them all. I want to also thank the members of the SW tools team for their support for this thesis. In addition, I want to thank the Technology Managers Jari Ahola and Jukka Perälä and Technology Director Pekka Silvennoinen from VTT for their

kind support and the possibility to get a leave of absence from VTT to finalise the dissertation.

I also want to thank Mark Woods who checked the language of this thesis.

Last but not least, I would like to express my warm and deep gratitude to my parents, sister, all friends, and especially Sara for great support, encouragement, and good moments. Your support has been invaluable.

Tampere, September 2007

Marko Palviainen

Contents

List of abbreviations

ADL	Architecture Description Language
Ajax	Asynchronous JavaScript and XML
AOP	Aspect-Oriented Programming
AP	Access Point
API	Application Programming Interface
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DCCI	Delivery Context Client Interface
CSD	Content Source Description
CSE	Context-Sensitive Element
DOM	Document Object Model
DSL	Domain Specific Language
DTD	Document Type Definition
GPRS	General Packet Radio Service
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
IRef	Instance Reference
J2SE	Java 2 Standard Edition
JVM	Java Virtual Machine
MIDP	Mobile Information Device Profile
MMS	Multimedia Messaging Service
MOP	Meta-Object Protocol
MVC	Model View Controller
OOP	Object-Oriented Programming
OS	Operating System
PC	Processing Context
PDA	Personal Digital Assistant
POD	Physical Object Description
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RSS	Really Simple Syndication
SFC	Suitability For Context

SI	Service Indication
SL	Service Load
SMIL	Synchronized Multimedia Integration Language
SMS	Short Message Service
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SVG	Scalable Vector Graphics
UDDI	Universal Discovery, Description, and Integration
UI	User Interface
UML	Uniform Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UMTS	Universal Mobile Telecommunications System
WAP	Wireless Application Protocol
WLAN	Wireless Local Area Network
WML	Wireless Markup Language
WSDL	Web Services Description Language
WWW	World Wide Web
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
XHTML	eXtensible HyperText Markup Language
XSLT	eXtensible Stylesheet Language Transformation

1. Introduction

Ubiquitous computing [Wei91, Wei93], also called pervasive computing, is a revolution driving adaptive computing systems [MSKC04]. It aims to make many computers available throughout the physical environment effectively invisible to the user by dissolving traditional boundaries for how, when, and where humans and computers interact [MSKC04]. Improving computer access to context increases the richness of communication in human-computer interaction and makes it possible to produce more useful computational services [Dey01]. This requires applications that respond to the requirements of the context at runtime and utilise current or predicted context information in dynamic adaptation.

Mobility brings a new dimension to the concept of context and context-awareness [DeA99]. Mobile and pervasive computing environments are heterogeneous and dynamic: everything from devices used, resources available, network bandwidths, to user context, can change drastically at runtime [BWL03]. Thus it is imperative for software systems and applications to be able to adapt to different computing environments and runtime changes appropriately [BWL03]. For example, the information about the mobile device and user activity, environment, other devices, location, and time can be utilised in different situations to enhance the interaction between the user and device [Kor05].

Mobile usage is spontaneous and applications should be fast to install, start, and use in mobile devices and wireless networks. The input and output capabilities and memory and processing power resources of mobile devices are typically limited in comparison to desktop computers. In addition, the wireless network connections offer typically less bandwidth than fixed line connections and may cause costs for the user. This all sets requirements for client and server-side adaptation methods that can provide more usable and efficient applications for specific users, contexts, and services available on the Web.

Browsers are a very generic way to implement User Interfaces (UIs) for various kinds of Internet services. However, many of the wired Internet services are difficult to use directly with standard browsers of mobile devices. For example,

the small display and keyboard of a mobile device set limitations for UIs and presentation of content. As well as the browsed contents, user agents and browsers should also be adapted for specific users, contexts, and services to provide a reasonable user experience.

This dissertation describes a task-based composition technique that is based on the content adaptation model of the World Wide Web Consortium (W3C) [Gim02] and discusses how tasks can be used in the dynamic composition of adaptive applications. The technique supports both automatic and user-directed adaptation and speculative adaptation in which the application parts are prepared for potential future contexts in the background. Tasks can dynamically prepare application instances for certain settings and contents coming from various sources and finally compose adaptive applications of the prepared instances.

Current dynamic adaptation techniques do adaptation primarily from scratch and do not take advantage of the performance gains, which can be achieved when re-using already adapted application elements [KPRS03]. The task-based composition technique facilitates the reuse of already adapted application elements and started adaptation tasks. In this dissertation it is argued that the task-based composition technique helps developers to construct adaptive applications for mobile devices and makes the dynamic composition of content and context-sensitive applications more fluent. This dissertation focuses on adaptive browsers and discusses how the task-based composition technique can support client-side dynamic composition of content and context-sensitive applications and improve performance when UIs are adapted for rapidly changing contexts and services available on the Web.

We respond to the following questions. How can the task-based composition technique be used? What are the true benefits and problems of the technique? How does it support dynamic composition of content and context-sensitive browsing applications? How does it support automatic, user-directed, or speculative adaptation in dynamic composition? How does it handle errors raised while a context-sensitive application is composed? These, and related questions are discussed in this dissertation.

This chapter is organized as follows. Dynamic composition techniques and adaptive applications and browsers are discussed briefly in Section 1.1. Then,

the problem statement of this dissertation is given in Section 1.2. The task-based composition technique approach is discussed briefly in Section 1.3. Finally, the main contributions of this dissertation are listed in Section 1.4.

1.1 Background

The background of this dissertation is introduced in more detail in the following subsections:

- *Mobile environment and adaptive applications.* The mobile environment sets various kinds of requirements for applications. Subsection 1.1.1.
- *Adaptive browsers.* Context-aware hypermedia can enable context-aware browsing, search, annotations, and linking applications [BCFH03]. Subsection 1.1.2.
- *Dynamic component-based composition*. The advantage of dynamic composition over static composition is that the behaviour of a new system will depend on object relationships being defined at runtime, instead of being defined and hard-coded in the files [DSGO02]. Subsection 1.1.3.

1.1.1 Mobile environment and adaptive mobile applications

In this dissertation a *mobile application* means a program executed in a mobile device that may possible utilize network connections.

The mobile environment poses great challenges for the development of applications (Figure 1). Applications are used any time and anywhere, and, thus, usage environment is different (e.g. more distractions) compared to a desktop PC. In contrast to desktop computers, mobile devices have typically limited input and output capabilities [KHL02], and restricted memory and processing power. In addition, wireless network connections typically offer less bandwidth than wired connections.

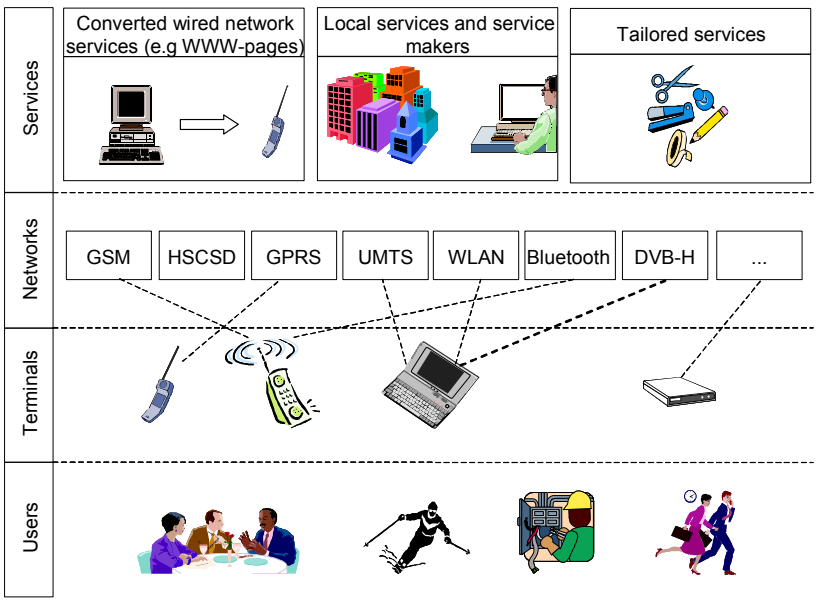


Figure 1. The number of applications and services is rising rapidly in the mobile environment. At the same time, the diversity of mobile users, terminals, and networks is increasing.

Mobile users are not accustomed to crashing devices and bugs are not considered acceptable. For example, correct memory handling is important, because mobile devices are typically turned on for long times (weeks or even months). To prevent memory leaks, the allocated memory should always be released after use.

The diversity of mobile devices and networks is increasing. Mobile devices will have multiple network connections and the available bandwidth may vary a great deal. In particular, applications must cope with the dynamics of the network and quality of service management, which is challenging due to the resource constraints of wireless devices and varying bandwidth [ITLS04].

To provide a reasonable user experience, applications should respond to the requirements of different kinds of users, contexts of use, networks, and services as well as be light, reusable, and transferable to various kinds of environments. At the same time, applications should demonstrate high quality and adaptability

to different kinds of environments. For example, adaptive and context-aware applications can support [SAW94]:

- *Proximate selection*, where the objects located nearby are emphasized in the user interface or otherwise made easier to choose.
- *Automatic contextual reconfiguration*, so that the component configuration and connections between these are changed if the context is changed.
- *Contextual information and commands*, which can produce different results according to the context in which they are issued.
- *Context-triggered actions*, so that simple IF-THEN rules specify how the context-aware system should adapt.

This dissertation uses terms defined by W3C [Lew05]. A *server* is a role adopted by an application when it is supplying a resource and a *client* is a role adopted by an application when it is retrieving resources. A *resource* is a network data object or service that can be identified by a *Uniform Resource Identifier* (URI) and may be available in multiple representations (e.g. multiple languages, data formats, size, and resolutions). The *delivery context* is a set of attributes characterizing the capabilities of the access mechanism and the preferences of the user [Lew03]. Delivery context can include information about the user (user profile and preferences), user agent, device, network, and the service itself. For example, Composite Capabilities/Preference Profiles (CC/PP) is a description of device capabilities and user preferences that is often referred to as a device's delivery context [Kis07]. Parts of the delivery context can be located in the Web. For example, to enable nomadic users to migrate seamlessly from one access device to another, user profile, and the state of the session should be stored in the network. Delivery Context Client Interfaces (DCCIs) are language neutral programming interfaces that provide Web applications access to a hierarchy of dynamic properties representing device capabilities, configurations, user preferences, and environmental conditions [WHR+07].

W3C has defined adaptation as a process of selection, generation, or modification producing one or more perceivable units in response to a requested Uniform Resource Identifier (URI) in a given delivery context [Lew05]. A requestor and an adaptor may act together as an element in the delivery path

providing a specific part of the adaptation [Gim02] (Figure 2). The requestor can modify the request and provide context information required for adapting the response appropriately.

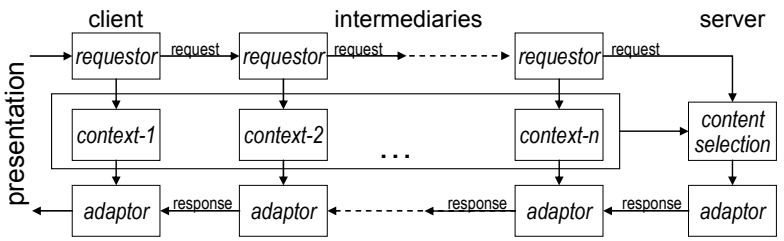


Figure 2. Adapting contents for different contexts [Gim02].

In this dissertation, an *adaptive application* is defined as a composite of co-operating components that are composed for a certain context. In addition, a *content and context-sensitive application* is understood to be an adaptive application that utilizes contents and may perform both selection and transformation actions for contents in different contexts. An *adaptive mobile application* means an adaptive application that is executed in a mobile device. Furthermore, *resource-aware applications* are adaptive applications that can reduce the quality of operation and thus consume fewer resources in order to provide an acceptable level of service despite the scarcity of resources [PSGS04].

1.1.2 Adaptive user agents and browsers

User Agent is a client within a device converting perceivable units into physical effects that the user can perceive and with which the user may be able to interact [Lew05]. A *browser* is an example of a user agent allowing the user to perceive and interact with information on the Web. User agents and mark-up languages are an easy way to transmit various kinds of information for the end-users. eXtensible Markup Language (XML) [BPS00] is a simple and very flexible text format for information exchange. XML is used in various kinds of Web applications and many mark-up languages (e.g. a Wireless Markup Language (WML), eXtensible HyperText Markup Language (XHTML), Synchronized Multimedia Integration Language (SMIL), Scalable Vector Graphics (SVG)

language, and Voice eXtensible Markup Language (VoiceXML) used in the mobile environment are based on XML.

In this dissertation *pull browsing* is defined as user initiated browsing, in which the client-side sends a request for the server-side that delivers the content in the response whereas in *push browsing*, the server-side can initiate the content delivery and notify the user about changes in the information (Figure 3).

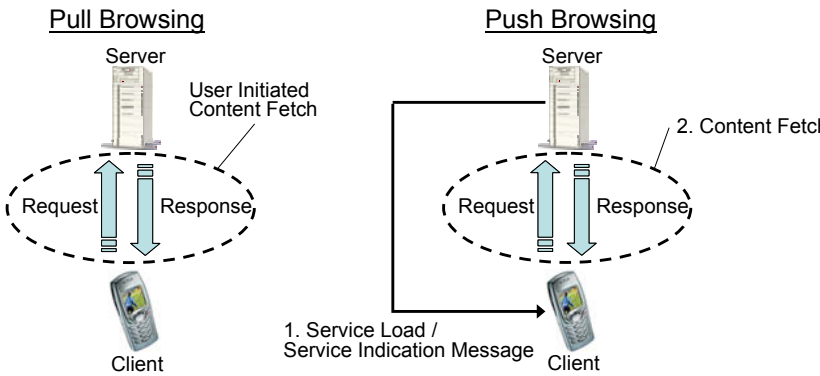


Figure 3. Browser-based user interfaces can be delivered to the user with pull and push browsing mechanisms.

For example, monitoring, remote control [HSH+01], and home automation applications have continuously changing information. This requires methods that update the changed information to the client-side. One solution is to do a *periodical refresh for the changing information.* For example, the *cache-control* header of HyperText Transfer Protocol (HTTP) and the *meta* (e.g. <*meta http-equiv="refresh" content="10"/*>) element of HyperText Markup Language (HTML) can be used for periodical refresh. *Really Simple Syndication (RSS) technology* [Win05] enables the owners of the Web sites to feed information about changes in the Web sites for users who have subscribed to these RSS feeds. As a result, a program known as a feed reader or aggregator can check RSS-enabled Web pages on behalf of a user and display any updated articles that it finds. The RSS formats provide Web content or summaries of the Web content together with links to the full versions of the content, and other meta-data. This information is delivered as an XML file, called an RSS feed.

However, the periodical refresh or information polling is an inappropriate mechanism for rapidly and irregularly changing information, because the user is not immediately notified of the changes. In addition, many unnecessary requests are made to the network. Highly interactive and real-time [SKK098] applications require more effective refresh methods. Combining the *push mechanism* and *instant messaging* [RHBK04] services into the browser can solve this problem (Figure 3). *Push* is a very powerful concept. It typically refers either to the mechanism or ability to receive and act on information asynchronously, as information becomes available, instead of forcing the application to use synchronous polling techniques that increase resource use or latency [Ort03].

Wireless Application Protocol (WAP) Push [WAPP01] allows content to be pushed to the mobile handset with Service Load (SL) and Service Indication (SI) type of messages. A WAP Push message can be delivered over WAP or Short Message Service (SMS) bearer. On receiving a WAP Push message, a WAP enabled handset will automatically enable the user to access the content referred in the message. The referred content can be illustrated with a user agent. For example, Multimedia Messaging Service (MMS) uses WAP push functionality to notify and deliver multimedia messages to the mobile device.

In general, the characteristics of the mobile environment set requirements for contents, browsers, and visualization. Browsing services designed for wired Internet are often hard to use directly with a standard browser in a mobile device. For example, tables or frames are difficult to present in the small screen of a mobile device. These usability problems can be partially solved by adapting contents on the client and server-side. Content adaptation can include both content selection and transformation actions. In addition, style sheets [BLLJ98] can redefine the presentation of the browsing content. The existing XML-based technologies enable context-aware wireless Internet services to adapt their content to a user's situation [PKP03]. For example, W3C has published a markup [LMF07] for general purpose content selection and filtering. The W3C's Document Object Model (DOM) Application Programming Interface (API) provides an XML tree-like document structure whereas eXtensible Stylesheet Language Transformations (XSLT) are able to generate e.g. WML or XHTML documents for display to the mobile user.

Contents can be adapted in the client, server, in one or more intermediate proxies or in all of these. A big part of content adaptation approaches focuses on server-side adaptation. For example, an approach that adapts HTML pages for mobile devices is proposed [KAK+00, LaH05]. Content items on a Web page can be transcoded into multiple resolution and modality versions so that they can be rendered on different devices [MSL99]. These content versions can be stored to InfoPyramid that enables a customizer to select the best versions of content items to optimally match the resources and capabilities of diverse client devices [MSL99]. In addition, a content adaptation system that takes into account the entire computing context and decides the best strategy for generating the optimal content version is proposed [LuL02]. The annotation semantic transcoding technique [NSS01] enables humans to associate external XML-based linguistic, commentary, and multimedia annotations with any element of any HTML document to facilitate machines to understand document contents and to make better transcodings for contents.

However, these previously described content adaptation approaches focus more on server-side content adaptation. The wireless connections typically offer less bandwidth than fixed line connections. In addition, the data transmission over wireless connections typically cost the end-user more. Browser-based Web applications typically require the user to submit a request to the server, wait for the server to process the request and generate a response, and then wait for the browser to update the UI with the results. This request-wait-response-wait pattern is extremely disruptive and lowers productivity [Smi06]. This all sets requirements for the client-side adaptation methods that can improve the usability, reduce costs, and lead to more intelligent and context-sensitive applications. We believe that the client and server-side adaptation methods together can provide the best user experience. However, this dissertation focuses on the client-side adaptation techniques and so for example, the server-side content adaptation techniques are not discussed in more detail.

It is not enough that only the browsed contents are adapted. In order to improve the usability, applications should be adaptable for specific contexts of use, users, devices, and services to include the required features only. Because the memory and processing capabilities of mobile devices are typically limited in comparison to desktop computers, the applications should be adapted to include only features needed in the browsing task in question. The ready-for-use time and

memory consumption is smaller, while the browser can be adapted to utilize device dependent APIs and external devices.

Adaptive browsers can offer specialised and context-sensitive controls and UIs for contents (e.g. for time tables) and services available on the Web. For example, a service requiring a good deal of textual input can be very laborious to use with the small keyboard of a mobile device. In order to minimize the use of the keyboard, the browser should offer e.g. textual templates (e.g. of contact information) for filling forms in a service available on the Web. In addition, a browser should be adaptable for attached external input (e.g. microphone or tag reader) and output (e.g. headset) devices during browsing. For example, if an external headset is attached, the browser could present the content vocally.

In this dissertation, browsers that can adapt for specific users, contexts, and services according to the delivery context [HeI01, Gim06], are called *adaptive browsers* (Figure 4). Furthermore, adaptive browsers that are used in a mobile device are called *adaptive mobile browsers*.

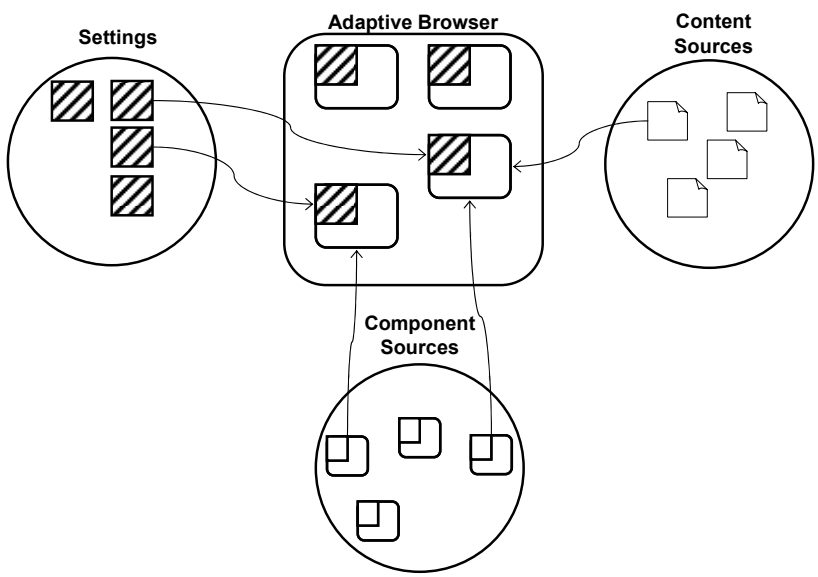


Figure 4. An adaptive browser.

An adaptive browser can be dynamically composed of components that are initialised for certain settings and contents coming from various sources. The correct components, settings, and contents are selected for the delivery context.

1.1.3 Dynamic component-based composition

The component-based techniques are widely used in software engineering. They promote a high level of abstraction in software design, implementation, deployment, and management. In addition, they facilitate flexible configuration (and, potentially, runtime reconfiguration), and foster third-party software reuse [CBG+04]. A substantial component-based software engineering community exists [Emm02] and numerous technologies support both standalone and distributed component-based application development [CBG+04]. For example, browser plug-ins [Moz01], Enterprise JavaBeans [Sun02], and the Common Object Request Broker Architecture (CORBA) Component Model [OMG02] support the development of component-based applications.

In software engineering, objects are runtime entities that are configured to achieve a certain end [NiM94]. Objects have identity, state, and behaviour, and are always runtime entities [Nie95]. Whether or not objects have their own internal threads, or may be otherwise considered "active," each object can be viewed as a kind of server, or *process* [NiM94].

Objects encapsulate services, whereas *components* are software abstractions that can be used in the construction of object-oriented systems [Nie95]. Components may also be runtime entities to which applications may connect [NiM94]. However, more generally, components must be composed and initialised before they are a part of a running application [NiM94]. This dissertation considers a component as an entity which cannot be directly modified.

With component *frameworks* [JoF88] the amount of programming effort and bugs can be significantly reduced. A component framework embodies policies and constraints that make sense in a particular functional domain [CBG+04]. Frameworks can offer interfaces and skeletons enabling developers to implement applications in a modular way so that certain points of the application can be replaced with solutions suitable for the usage environment. At the same time, existing and well-tested solutions can be reused in various kinds of

configurations. The modular structure is easier to understand, the development work (e.g. analysis, design, implementation, and testing) can be organized better, and developers can become acquainted with the application by studying well-defined interfaces and components. However, the modular structure can cause computational overhead. Interfaces require memory and individual components may contain methods and code not needed in all configurations.

The *configuration* – or *composition* – itself need not correspond precisely to the object-level view, since both more fine-grained and more coarsely grained components may come into play [NiM94]. Modules, packages, frameworks, and even generic configurations or architectures are also good candidates for more coarsely-grained components [Bra92].

Component-based composition requires components conforming to *architectural styles* [ShG96] that determine the *plugs* each component may have (i.e., exported and imported *services*), the *connectors* that may be used to compose them, and the *rules* governing their composition [AcN01]. The components can have various kinds of dependencies to other components, which can complicate component-based composition. The composition of component-based applications can be facilitated with frameworks, which may offer skeletons, interfaces, and ready-made components for the applications of a specific domain.

A virtually unlimited set of functionalities and services can be composed of a set of basic service components [Men00]. Components can be selected for the context, fetched from the Web, and composed into various configurations resulting in a highly tailored application that can, for example, include alternative UIs, controls, resources, and services. However, in contrast to static composition, *dynamic composition* requires an infrastructure and may increase complexity and overhead. Methods that can find components and then select the most suitable components for the application are needed. Also, remote components and resources should be possible to use in composition. Then, methods that can construct application instances of the components configured for the context are needed. Finally, methods that are capable of composing the actual component-based application of the prepared instances are required.

Component reconfiguration and *dynamic re-composition* are commonly used application-transparent adaptations in component-oriented applications. The components may provide special interfaces allowing certain parameters of the components to be changed. The dynamic re-composition of application components may involve addition, removal and/or replacement of the components at runtime [MuG05]. In the case of a dynamic replacement the state of an expendable component must be transferred to its successor. In addition, the integrity of the interactions ongoing at the time of dynamic re-composition must be maintained in order to ensure the consistency of the application [MuG05].

1.2 Problem statement

The collection and analysis of context information, decision of the adaptation actions, and triggering for the decided adaptation are needed in order to adapt an application for the context [ATB04]. Interacting with a variety of sensors to capture the context, interpreting the measured values to the desired format, and using them in a meaningful way may cause a large development overhead [ChK00]. Thus it is important to separate the application from the actual context sensing part. As a result, it may be possible to utilize the same context-sensing system in various context-aware applications.

Mobile usage is spontaneous and applications should be fast to install, start, and use in devices with limited memory and processing power. In addition, wireless network connections offer typically less bandwidth than fixed line connections and may cause costs for the user. The context can change rapidly when the user is moving in the physical environment. This all requires methods that can make the dynamic composition of adaptive applications more fluent.

The focus of this dissertation is on the execution of triggered adaptation actions. More precisely, this dissertation discusses how it is possible to help developers to construct adaptive applications for mobile devices and to make the dynamic composition of content and context-sensitive applications more fluent.

More fluent composition requires a composition technique that can speed up and reduce adaptation costs and enable end-users to control the adaptation. The composition technique should help developers to implement adaptive

applications that are able utilise previously-started adaptation actions and prepared content and context-sensitive application instances in adaptations. Errors may arise while an adaptive application is composed for new contexts. Thus context-sensitive methods are needed for error handling. New kinds of requirements may emerge for an adaptive application. Thus it is important that the composition technique does not provide only a fixed set of methods supporting adaptation but that it should also be possible to extend the technique with new adaptation methods. In addition, it should be possible to change the adaptation strategies at runtime.

This dissertation addresses the following problems related to the dynamic composition of adaptive mobile applications:

- *Supporting active and passive context-awareness in dynamic composition of adaptive applications*. How to support active and passive context-awareness in dynamic composition? Subsection 1.2.1.
- *Context-sensitive handling of errors appeared in dynamic composition of adaptive applications*. How to handle errors raised while an adaptive application is composed? Subsection 1.2.2.
- *Supporting speculative adaptation in dynamic composition*. How to compose an application for possible forthcoming contexts? Subsection 1.2.3.
- *Supporting client-side dynamic composition of adaptive mobile browsers*. What are the requirements for adaptive mobile browsers? Subsection 1.2.4.
- *General quality goals.* What are the general quality goals of a dynamic composition technique supporting composition of adaptive mobile applications? Subsection 1.2.5.

This dissertation focuses on utilisation of the current and predicted context information in the dynamic composition of adaptive applications. The context acquiring, processing, and predicting methods are not directly in the scope of this dissertation and so are not discussed in detail. Instead, this dissertation assumes that the current or predicted context information is available for an application. One of the main goals of this dissertation is to describe how

dynamic composition can be used in content and context-sensitive applications and how the composition process can be improved on the client-side.

1.2.1 Supporting active and passive context-awareness in dynamic composition

The key to context-awareness is in doing it behind the scenes [AAH+97]. The application can perform *internal adaptations*, in which the context and knowledge about the task of the user may guide application adaptation [RBH04]. Parts of an application that are not needed at the time can be removed, which saves resources. Optional components, for example, can be initialised only when a user is in a resource-rich environment. However, it is important that context-aware applications do not cause embarrassing or dangerous situations for the users and are predictable and understandable to use. For example, it should be easy to set user preferences.

Context-aware computing can be active or passive [ChK00]. *Active context awareness* means that an application automatically adapts to discovered context, by changing the behaviour of the application [ChK00]. Active context-aware computing can help to eliminate unnecessary user cooperation and make technology as "calm" as possible [ChK00]. *Passive context awareness* means that an application presents the new or updated context to an interested user or makes the context persistent for the user to later retrieve [ChK00]. Active context influences the behaviour of an application, while the passive context can be relevant but not critical for the application. For example, an automatic call forwarding system can use the location information of the user actively, while the Active Map application [Wei93] can utilize it passively [ChK00].

Active context-aware information push may cause usability problems [CMD01]. If the user is currently engaged in reading, it may become overwritten. The problem can be solved with a back button that enables the user to return to the previous context in the application or with a hold feature, which enables displayed information to remain on the screen despite any changes in the context [CMD01].

Users are tolerant of frequent changes between fidelities with small perceptual differences, but are intolerant of frequent, perceptually large changes [Nob00].

This sets requirements for techniques that enable developers to compose fine-grained adaptations for applications.

The composition technique should support both active and passive context-awareness in the composition of adaptive applications. For example, it should enable end-users to control the adaptation. In addition, it should offer feedback for end-users about the progress of the adaptation.

1.2.2 Context-sensitive handling of errors appeared in dynamic composition

It is useful for programming on a large scale and for application testing, if a language has features to explicitly cope with errors and exceptions [Sch99]. Errors may arise while an adaptive application is composed and all the needed adaptations cannot be always necessarily successfully executed. For example, if a mobile device is disconnected, it is not possible to fetch contents from the Web. To prevent deadlocks, it should be possible to define time-outs for adaptations and recovery mechanisms for fault situations.

An adaptive application requires context-sensitive methods for error handling. For example, context-sensitive error messages and UIs can be displayed to enable the user to direct adaptation in fault situations.

The dynamic composition technique should enable developers to define fault tolerant adaptations for applications and context-sensitive methods for both automatic and user-directed error handling.

1.2.3 Supporting speculative adaptation in dynamic composition

Adaptive systems must accommodate high-dimensional sensory data, continue to learn from new experience, and take advantage of new adaptations as they become available [MSKC04]. For example, contextual information can be captured for identifying which locations the user has visited [AAH+97]. Unfortunately, even if context history is generally believed to be useful, it is rarely utilized in context-aware applications [ChK00].

A context-aware application should learn about user behaviour in order to improve the context awareness [LKAA96]. Users may have certain behavioural routines. For example, they may often do same tasks at the same time of the day and use the same routes for moving inside buildings. In order to make adaptation more fluent, an adaptive application should learn user behaviour, sense the activity of the user, and use the idle time of the application for *speculative adaptation* that prepares application parts for potential future contexts in the background.

For example, the user interfaces of a home automation system can be prepared in the background for the different locations. As a result, they can be fast provided for the user moving at home. Web pages the user may visit in the near future can be prefetched [FiS03]. For example, a browsed document can provide a set of pre-fetching hints enabling the browser to prefetch and store specified documents in its cache in the background. Furthermore, the referenced pages from hyperlinks embedded in a browsed object can be prefetched to the cache [ChY97, Duc99]. Later, when needed, prefetched documents can be fast illustrated for the user. Speculative adaptation can also be used to *automated hoarding* [JHE99] in which non-local files are pre-fetched to the client cache prior to disconnection.

In addition, pre-resolving, pre-connecting, and pre-warming techniques can decrease network latencies [CoK02]. *Pre-resolving* can perform the DNS query prior to the user requests and eliminate the long query time from the user perceived latency whereas *pre-connecting* can establish a TCP connection prior to the user request. Finally, in *pre-warming* the client can send dummy HTTP requests prior to an actual request and so the server is in a warm state and e.g. the results of a reverse-DNS query are cached to the server.

The dynamic composition technique should help developers to implement speculative adaptations for content and context-sensitive applications. Firstly, it should facilitate utilisation of various kinds of methods that can predict potential future contexts. Secondly, it should make it easier to start speculative adaptations for predicted contexts, to utilise previously started adaptation actions and prepared content and context-sensitive application instances in speculative adaptations, and to recognize and stop those started adaptation actions that are not suitable for the current or predicted contexts.

1.2.4 Dynamic client-side composition of adaptive mobile browsers

In a mobile device, it can be difficult to navigate between separate applications. Browsing services are needed in many applications. For example, contact information (e.g. for a phone number) that is available on the Web may be needed in an application running on a mobile device. Browsers embedded in adaptive mobile applications offering Internet services directly and fast available for the user could substantially improve usability. For example, it would be useful if a Web browser could be integrated into a context-aware tourist guide application [AAH+97].

A user agent or browser should respond to the requirements of the mobile environment. The generic requirements collected from different sources identified for browsers are enumerated below. Firstly, the mobile browser should offer navigation operations as commonly found in browsers of the wired Web, e.g. navigate a new page, navigate in history (back, forward), refresh browsed content (reload) and abort the navigation (stop) [LNR96]. At the same time, it should support different kinds of communication protocols, connections, and authentication methods. In addition, the use of the wireless network should be optimized. For example, this can be made with caching and by adding concurrency [Wat94].

New and emerging applications such as push and instant messaging (where the content delivery is initiated outside of the browser) and context-aware services (in general) bring new challenges how to support browsing within the application or when using a separate browser. Much information and many services are relevant only in a limited context [HSP+03]. In order to improve usability, new user interfaces, navigation, content visualization, contextual sensing, resource discovery, and augmentation methods for interaction with the environment [LaH05, Gri04, KSKB02] are needed. Particularly, an adaptive browser should support:

- specialized, adaptive user interfaces [HeI01],
- combining various kinds of content types to work together [KSKB02, PaL05b],

- the client-side adaptation, and the delivery of the context information for the server-side adaptation [BGGW02],
- modal and non-modal controls in browsing [KSKB02],
- utilizing various resources (i.e., user agent profile, stylesheets, and user preferences) and services in illustrating the browsed content in a specific way [LaH05], and
- extensibility with new browser components [Men00].

Composing an adaptive browser from scratch requires a lot of effort and knowledge. Techniques that facilitate the dynamic composition of adaptive mobile browsers that can be embedded in context-aware applications are needed. This dissertation focuses on adaptive mobile browsers and discusses how the task-based composition technique can support client-side composition of content and context-sensitive applications and improve performance when UIs are adapted for rapidly changing contexts and services available on the Web.

1.2.5 General quality goals

A dynamic composition technique must provide common services and functionality for component-based composition. For example, it should enable developers to define composition schemas that can configure application components for a rapidly changing context. The following list summarizes the general requirements for the dynamic composition technique. The task-based composition technique itself is evaluated against these general quality goals in Section 9.1:

- **Generic**. To be generally applicable, the technique should be generic so as to be usable in the dynamic composition of a wide range of applications.
- **Extensible**. As adaptation is needed in different kinds of applications and environments, the technique must be extensible. The technique cannot only provide a predetermined and fixed set of methods supporting dynamic composition; instead it must allow the developers to extend it with new kind of methods supporting dynamic composition of various kinds of adaptive applications.

- **Policy independence**. A general-purpose component-based systems building technology should offer generic mechanisms; it should not prescribe policies, constraints, services, or facilities that are specific to particular application domains or deployment environments [CBG+04].
- **Scalable.** The technique should be scalable to support adaptation in very different kinds of environments and devices offering processing and memory capabilities of a different level. For example, the technique should support both client and server-side adaptation. For making the technique applicable also in mobile devices, the implementation of the technique should have a small memory footprint and offer language and policy-independent methods [CBG+04].
- **Separation of concerns**. The technique should promote a clear separation of concerns between the application's functional code and the adaptation-specific code.
- **Incremental**. The technique should support the reuse of composition schemas. For example, it should be possible to utilise other composition schemas in a composition schema.
- **High performance.** The technique should not cause an inherent performance cost and overhead that will significantly disturb the user of an adaptive application.

1.3 Approach: Task-based composition of adaptive applications

It is important to raise the abstraction level of component-based composition in order to make it is easier for developers to implement adaptive applications for mobile devices. If the dynamic composition concerns can be separated from the rest of the application, it is easier to customize separate composition schemas to satisfy the application requirements. In addition, it may be possible to reuse separate composition schemas in different applications.

An interpreted task-based composition language enables composition strategies to be changed at runtime. However, in the name of simplicity, aspect specific languages do not typically offer all the structures and the conditional and

iterative commands that the general-purpose languages like Java and C++ provide [LBS+98]. This lack of programming constructs can be compensated for by allowing the programmer to use, for instance, Java or C++ code in the composition schemas [LBS+98]. For example, an XML-based aspect language for component composition can have parts invoking methods of plugins implemented e.g. with Java or C++.

The content adaptation model of W3C is a solution where a requestor and an adaptor can act together as an element in the delivery path providing a specific part of the adaptation [Gim02]. The requestor can modify the request and provide the context information required for adapting the response appropriately. The *task-based composition technique* [Pal05, Pal07] is based on the content adaptation model of W3C and supports dynamic composition of component-based applications. Tasks can have various kinds of adaptation actions that can compose application instances, request other tasks, adapt the received responses, and deliver responses for the requestors of the task.

We believe that the *task-based composition technique* can help developers to implement adaptive content and context-sensitive applications for mobile devices. Tasks are configured to a *task factory* that composes tasks for adaptation requests. The task configuration is defined with an XML-based language that enables developers to describe tasks and context-sensitive adaptation actions and settings elements for them. Settings elements configure an action to work in various kinds of application environments. For example, they can configure actions to fetch application instances from certain sources and to add them to specific targets. In addition, settings elements can also define context-sensitive parameters and content elements for adaptation actions. Tasks can prepare content and context-sensitive application instances in many phases and finally compose an application of these. Both context-sensitive tasks and application instances can be cached, which can speed up adaptation of content and context-sensitive applications.

Based on the gained experiences with the task-based composition technique and different case studies, this dissertation outlines the characteristics of a generic framework that supports dynamic composition of content and context-sensitive applications. The framework separates composition concerns of the rest of the

application and offers tasks and actions implementations that facilitate dynamic composition of content and context-sensitive applications (Figure 5).

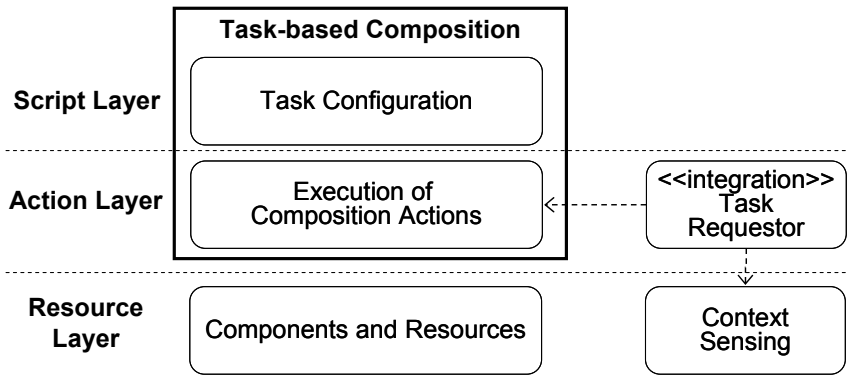


Figure 5. Task-based composition of adaptive applications.

The task requestor works as an integration element that can observe the context and request adaptation tasks to adapt the application for new contexts, if the specific parts of the context are changed.

Different task configurations can be constructed to support the dynamic composition of various kinds of applications. For example, we constructed a small client-server application, in which both the client and server-side were implemented with tasks [ImP07]. As a platform, the system provides a common foundation to implement various kinds of adaptive applications that can be composed with asynchronous and synchronous tasks and which enables the end-user to control the adaptation process. In addition, the platform offers context-sensitive handling for errors raised while an application is composed.

The MIMEFrame framework [PaL06] defines architecture for adaptive user agents and browsers (see Chapter 7). However, although MIMEFrame and the task-based composition technique facilitate implementation of adaptive browsers, it may be a too difficult task for developers that do not have a lot of programming experience to construct totally new adaptive browsers. However, if a lot of ready-made browser components and composition schemas are available, it is possible to implement new kinds of browsers without manual coding. Only composition schemas have to be edited. This can be made with standard text or

XML editors. In addition, the mPlaton editor (see Section 6.5) facilitates implementation of the composition schemas.

1.4 Contributions

The main contributions of this dissertation are the following:

- *A general task-based composition technique that facilitates dynamic composition of adaptive mobile applications.* The author has developed the task-based composition technique that is based on the content-adaptation model defined by W3C. It is an integrable and universal approach that can be utilised in a wide range of adaptive applications (Chapter 4).
- *The MIMEFrame framework that defines architecture for adaptive user agents and browsers.* It helps developers to implement very extendable user agents for mobile devices that can offer specialized user interfaces for almost any kind of Web application and that can be embedded in other applications. The author has made Java implementations for the MIMEFrame framework (Chapter 7).

The task-based composition technique is evaluated with case studies. Related to this, the author has carried out:

- *The reference implementation of the task-based composition technique.* The author has made a reference Java implementation for the technique and carried out the different case studies to analyse the technique (Chapters 4, 5, 6, and 7).
- *A tool for the task-based composition technique.* We have previously implemented a framework (called FEdXML) to facilitate implementation of component-based XML editors [PaL05c]. The author has utilised FEdXML and implemented a specialised XML editor helping developers to implement task configurations (Section 6.5).
- *The analysis of case studies that were implemented for mobile devices.* The task-based composition technique is evaluated in the domain of

adaptive browsers. The author has implemented three case studies for mobile devices and evaluated them (Chapter 7).

As a background, context-awareness and adaptation are discussed first in Chapter 2. Then, dynamic component-based adaptation techniques are discussed in Chapter 3. The task-based composition technique and abstract usage scenarios for it are described in Chapter 4. The usage scenarios describe the main functional requirements for case studies that utilise the task-based composition technique in the domain of content and context-sensitive applications. The mobile devices set requirements for the implementation of the task-based composition technique. Implementation issues are discussed in Chapter 5. The reference implementation of the task-based technique, called TaskCAD, is discussed in Chapter 6. Then, Chapter 7 presents the case studies that discuss the quality requirements of the task-based composition technique. The case studies are based on the usage scenarios described in Chapter 4. Comparisons to related work are given in Chapter 8. Finally, conclusions are drawn in Chapter 9.

The parts of this thesis are already discussed in the following publications:

1. The first version of the MIMEFrame framework was described in publication [PaL03].
2. The first version of the task-based composition technique was discussed in the paper [Pal05].
3. The paper [PaL05b] presents how modular and generative approaches and the task-based composition technique can be used in the implementation of adaptive mobile browsers.
4. The paper [PaL05c] discusses a FEdXML framework that facilitates implementation of specialised XML editors.
5. The paper [PaL06] describes how tasks can dynamically compose adaptive browsers of the MIMEFrame components.
6. The paper [Pal07] discusses task-based composition of the context-sensitive UIs of physical environments.
7. In the paper [ImP07], a small client-server application that is implemented with tasks is a test case for a new reliability testing approach.

2. Adaptation of context-aware applications

The concepts of context and context-aware computing are central throughout this dissertation. Before adaptation techniques can be discussed, it must be understood what the concept of context and context-aware computing means. In addition, adaptation of applications requires techniques and infrastructures that are able to sense and process context information and to predict potential future contexts. Finally, methods that are able to adapt the application for the sensed or predicted context are needed. This chapter is organized as follows. The concepts of context and context-aware computing are discussed briefly in Section 2.1. Then, classifications for adaptation are given in Section 2.2. Finally, adaptation techniques are discussed in Section 2.3.

2.1 Context and context-aware computing

Since Mark Weiser presented the idea of the computer of the 21st century [Wei91], adaptive and context-aware applications have been researched a lot. The concept of context is characterized in many sources. For example, context is defined to be the identities of nearby people and objects, and changes of those objects [ScT94]. Context can express relationships between people, places, and things with predicates defining identity, location, activity, and time [KBM+02]. In other words, the important aspects of context are: where you are, who you are with, and what resources are nearby [SAW94]. Context is also defined to be a subset of physical and conceptual states of interest to a particular entity [Pas97]. This dissertation uses a more general context definition of [Dey01] defining that:

”Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.”

In other words, if a piece of information can be used to characterize the situation of a participant in an interaction, then that information is a context. Context can be further divided into computing, user, and physical context categories [SAW94]. Time can be the fourth context category [ChK00].

High-level context can define the current activity of the user [ChK00]. It can be recognized e.g. by using machine vision, or calendar directory to find out what the user is supposed to do at a certain time, or by combining several simple low-level sensors [SBG99]. For example, in the TEA project [SBG99] high-level contexts were recognized with a neural network utilizing different kinds of sensors.

Context-aware computing is the ability of applications to discover and react to the changes in the environment in which, a mobile user is situated [ScT94]. More precisely, a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the task of the user [Dey01].

Context-aware features can be classified into *contextual sensing*, *adaptation*, *resource discovery*, and *augmentation* features [Pas98]. Contextual sensing features are needed for adapting applications for the context whereas resource discovery features enable a computer to discover other resources within the same context as itself and exploit these resources while they remain in the same context. Contextual augmentation features enable digital data to be associated with a particular context that it is related to. The context sensing, resource discovery, and augmentation features are not directly in the scope of this dissertation and so are not discussed in greater detail.

2.2 Classifications for adaptation

Self-adaptive software modifies its own structure and behaviour in response to changes in its operating environment [OGT+99].

Adaptation can be general or application-specific. In some cases the information exchanged by components is adapted [FGC98, Nob00], rather than the application components themselves [RoR05]. The client, server, or one or more intermediate proxies, or all of these may do that kind of adaptation [BFK+00].

Adaptation of software can be carried out across a broad scale: at one extreme, *macro adaptation* can be achieved only by extensively re-engineering the system; at the other, *micro-adaptation* consists of fine-tuning of running

software [VKK01]. No clear tendency with regards to the scope of adaptation exist [RoR05]. Some approaches adapt the application as a whole [AzJ00, FGC98, Nob00, BCA+00, CEM03, ChK00], while others include the ability to adapt discrete application components [SeA00, GCB+00, BHL01, MUCR02, SEK03, VZL+98, VTA04, MCR04]. The latter approach is more flexible since it offers more adaptation options and finer granularity, but is more complex to implement since it requires a more sophisticated implementation mechanism [RoR05].

The adaptation techniques can be classified in many ways. A few of the major classifications for adaptation techniques are introduced in the following subsections.

2.2.1 Static and dynamic adaptation

Adaptation can be divided into static or dynamic adaptation (Figure 6) [SaM03]. Static adaptation is made before using the application while dynamic adaptation is done at runtime. Dynamic adaptation can involve complex issues such as managing adaptation of software components that are used simultaneously by applications with different (and possibly conflicting) requirements, and maintaining a consistent external view of a component, the behaviour of which evolves over time [HIR01].

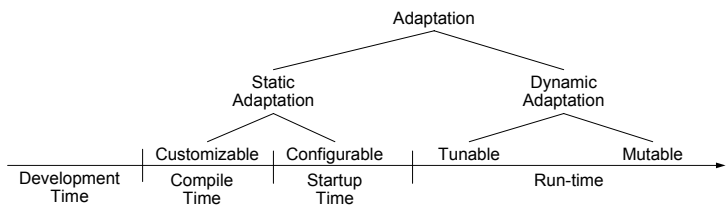


Figure 6. Taxonomy for static and dynamic adaptations [SaM03].

Tunable software does not support the modification of the code of business logic but supports fine-tuning of crosscutting concerns in response to changing environmental conditions [MSKC04]. For example, AspectIX enables runtime tuning of the distribution behaviour of Common Object Request Broker Architecture (CORBA) [OMG06] applications.

Mutable software enables a composer to change even the imperative function of a program, enabling dynamic re-composition of a running program into one that is functionally different. Languages like CLOS [ABB+89] and Python [Ros96] provide direct support for dynamic re-composition. Existing languages (e.g. Java and C++) can also be extended to support new keywords and constructs enhancing the expressiveness of an adaptive code [MSKC04].

2.2.2 Laissez-faire, application-aware, and application-transparent adaptation

The adaptation can be classified into *laissez-faire, application-aware,* and *application-transparent* adaptations [JHE99].

In the *laissez-faire adaptation,* an application triggers and implements all adaptations [JHE99]. Laissez-faire adaptation allows individual applications to adapt precisely according to their own goals, and does not require any operating system support [NoS99]. However, laissez-faire adaptation cannot support concurrency [NoS99]. Concurrent applications that use laissez-faire adaptive schemes will each adapt to the same set of environmental changes, and compete for the same set of scarce resources [NoS99]. Without some central authority, they are likely to interfere with one another and adapt at cross purposes.

In *application-transparent adaptation*, the burden of adaptation is entirely borne by the system [NoS95]. Applications explicitly interact with middleware services that both trigger and execute adaptations independently of the application. This type of adaptation is the most appealing from the perspective of the software developer, since the application can be implemented using conventional techniques while still realising the potential benefits of adaptation [RoR05]. In addition, this approach is completely backward-compatible and existing applications continue to work even when mobile [NoS95]. Unfortunately, application-transparent adaptation is difficult to achieve in practice [RoR05].

Application-aware adaptation [NoS95] may have features of both laissez-faire and application-transparent adaptations. In application-aware adaptation, applications can decide how to best adapt to the changing environment while the system can provide support through the monitoring of resources and the enforcing of resource allocation decisions [JHE99].

2.2.3 Reactive and proactive adaptation

In *reactive adaptation* [Cap03], an application can specify what portion of context it is interested in. As a result, it is notified when specific parts of context are changed. *Proactive adaptation* refers to the ability of the application to deliver the same service in different ways when requested in different contexts and at different points in time [Cap03].

For example, in a publish-subscribe architecture [WeB98] applications can register their interest in particular context changes, and then an event delivery mechanism notifies registered applications of relevant changes happening in the environment. The set of events that can be detected and delivered is extensible and can be dynamically modified by the application. It is then entirely up to the application to decide what to do (i.e., how to adapt) once it is notified about these changes.

Odyssey [Sat96] is a platform for mobile data access that extends the publish-subscribe mechanism by enabling applications with primitives to register the behaviours that the system should automatically invoke when specific context configurations are entered. In Odyssey, applications can register an interest in particular resources by defining the acceptable upper and lower bounds on the availability of that resource, and by registering an `up-call procedure' that must be invoked whenever the availability of the resource falls outside the window of acceptance. A *viceroy* component monitors resource usage, uses the registered up-calls, and notifies applications about significant changes. When an application is notified of a change in resource availability, it must adapt its access. *Warden* components implement the access methods on objects of their type: they provide customised data access behaviour (e.g. different replication policies) according to type-specific knowledge.

Gaia [RHC+02] offers a more general approach to reactive adaptation to context changes, as it does not focus on one particular service (data access). In Gaia, physical spaces and their ubiquitous computing devices are converted into active spaces. Gaia adapts application requirements to the properties of its associated active space, without the application having to explicitly deal with the particular characteristics of every possible physical space where they can be executed. An active space hides the complexities of dealing with heterogeneous devices and

sensors, and provides a generic interface that allows application engineers to interact with any physical space in a uniform way.

A common limitation of publish-subscribe, Odyssey, and Gaia approaches is the lack of support for *proactive adaptation* to context changes [Cap03]. More precisely, these techniques do not provide mechanisms that facilitate the customisation of the services the application delivers to its user, based on context.

Carisma is a middleware for context-aware applications [Cap03]. It is based on reflection and metadata and supports both reactive and proactive application adaptation. Adaptation takes place by means of metadata, or application profiles, that contain both reactive and proactive associations. Through reflection, applications can dynamically alter meta-information, thus adapting their behaviour to varying context conditions and user needs.

2.2.4 Speculative adaptation

An adaptive application can learn about user behaviour, sense the activity of the user, and use the idle time of the application for speculative adaptation that prepares application parts for the potential future contexts in the background. Network connections can be opened, information can be pre-fetched, and application instances can be prepared for the potential future contexts in the background. As a result, it is faster to adapt an application for the new context if the opened network connections and prepared parts can be utilised in adaptation.

Speculative adaptation requires methods that are capable of predicting potential future contexts. For example, information pre-fetching requires models capable of predicting which contents should be downloaded to the cache. The prediction models can use information about the previous behaviour of the user and try to predict what the user will possibly do next. If the prediction fails, the failing must be recognized, the speculative adaptations withdrawn, and the prediction model updated to improve future prediction accuracy [PBT+04].

Prediction methods are not directly in the scope of this dissertation and are therefore not discussed in detail. Only a few of them are discussed briefly in the following paragraphs.

For example, Web logs mining is used for predicting the path of the Web surfer [PiP99, SYLZ00, YZL01]. The neural networks [KrS93] are used in Adaptive House project [Moz98] to predict the next location and activities of the user. A method [AsS03] that is based on Markov models [Rev76] can predict the user's future movements. In addition, an approach [Kat02] applies Hidden Markov Models [Rab89] and Bayesian Networks [Cha91] to predict people's movement.

Mobile motion prediction algorithms (MMPs) [LiM96] to predict the next location of a mobile user according to the user's movement history are also proposed. An MMP algorithm consists of regularity detection and motion prediction algorithms. The motion prediction algorithm uses the database of regular movement patterns and random probability information with constitutional constraints to predict the next movement track of the mobile user. The Movement Circle (MC) and Movement Track (MT) models extend the Markov model for modelling the movement of mobile users. The MC algorithm is based on the closed circuits-like model of user behaviours and is used for predicting long-term regular movement whereas the MT algorithm is based on the MT model and is able to predict the routine movement of the user. The prediction accuracy ratio is over 50% if a user has 70% or less randomness in his or her movement [LiM96]. If the user movement randomness is 30% or less, the prediction accuracy ratio is more than 75%.

A prediction model can also base on evolutionary algorithms [BCS03]. For example, the population of finite state machines (prediction machines) may provide a prediction model for the next Web requests of the user at each session [BCS03]. The best machine in the population is used to make predictions, and at the end of each session a number of the machines are mutated, evaluated, and a new population is computed, taking into account the information provided by the last session. As a result, the proposed prediction system can react quickly to changes in the user's habits or in site structure.

2.3 Key techniques for dynamic adaptation

Dynamic adaptation requires programming paradigms supporting adaptation of both functional and nonfunctional system properties [MSKC04]. For example, object migration [GCB+00, BHL01], the selection of alternative methods [ChK00,

VZL+98], the substitution of object implementations [SeA00, MUCR02, MCR04], and middleware reconfiguration [AzJ00, BCA+00, CEM03, VTA04, RoR05] mechanisms have been used in adaptation.

The dynamic adaptation techniques can be divided into parametric and compositional adaptation techniques [MSKC04] (Figure 7). The parameter adaptation modifies variables determining the behaviour of a program, while compositional adaptation exchanges algorithmic or structural system components with others adapting the program to better fit to its environment.

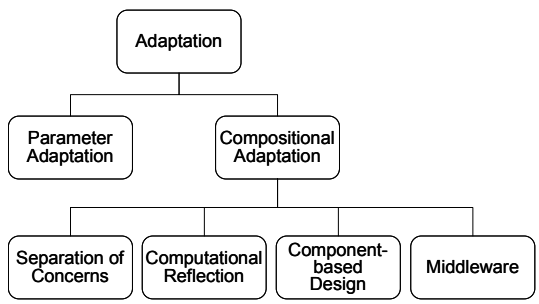


Figure 7. Key techniques for adaptation [MSKC04].

Separation of concerns, computational reflection, component-based design, and middleware solutions are the key techniques for reconfigurable software design and compositional adaptation [MSKC04]. These are discussed briefly in the following subsections.

2.3.1 Separation of concerns

Separation of concerns has become an important principle in software engineering [CzE00]. Presently, the most widely used approach appears to be Aspect-Oriented Programming (AOP) [Kic96, MSKC04]. AOP provides abstraction techniques and language constructs to manage crosscutting concerns. The code implementing these concerns, called aspects, is developed separately from other parts of the system. In AOP, pointcuts are sets of locations in the code where the developer can weave in aspects. A specialized compiler, called an aspect weaver, can combine different aspects into functional code. For example, separately developed adaptation behaviour can be woven into applications.

If an adaptive program is composed statically at development time, then any adaptive behaviour is hardwired into the program and cannot be changed without recoding and recompilation [MSKC04].

For example, AspectJ [KHH+01] and Compositional Filters [BeA01] can weave adaptive behaviour into existing applications at compile time. In contrast, TRAP/J [SMCS04] weaves generic interception hooks into application code during compile time. Afterwards, the composer weaves new adaptive components into application at runtime. Finally, a meta-object protocol uses reflection to forward intercepted operations to the adaptive components.

2.3.2 Compositional reflection

The *computational reflection* [Smi82, Mae87] refers to the ability of a program to reason about, and possibly alter its behaviour by enabling the system to reveal selected details of its implementation without compromising portability [MSKC04]. As a result of such introspective processing, a reflective system can inspect and change itself during the course of its execution. The major drawbacks of reflection are an increased performance overhead and the necessity to maintain integrity of the system [BlC97].

In a computational reflection, an application is typically structured at a base-level, which deals with application concerns, and a meta-level, which deals with reflective computation [BIC97]. The base-level functionality of a program is typically augmented with one or more meta-levels, each of which observes and manipulates the base level [KMSS02].

Reflection can be divided into structural and behavioral reflection defined thus:

- *Structural reflection*. The ability of a language (or system) to provide a complete reification of the program currently executing, for instance, in terms of its methods and state. This enables the programmer to inspect or change the functionality of the program and the way it models the domain. For example, structural reflection can address issues related to class hierarchy, object interconnection, and data types [MSKC04].
- *Behavioural reflection* focuses on the computational semantics of an application [MSKC04]. It is the ability of a language (or system) to

provide a complete representation of its own semantics, in terms of internal aspects of its runtime environment [CEM03]. This enables the programmer to inspect or change the way the underlying environment processes the program, for example, with regard to non-functional properties and resource management.

The *meta-object and Meta-Object Protocol (MOP)* interfaces enable introspection and intercession of the base-level objects by supporting either structural or behavioral reflection. Meta-object protocols are used in many language-oriented techniques that support dynamic adaptation of application. For example, MOP mechanism is used in Adaptive Java, ARCAD, TRAP/J, Kava, and R-Java.

Adaptive Java [KMSS02] extends Java language with constructs supporting computational reflection. It encapsulates application components inside their meta-level objects with the wrapper pattern. As a result, an application can be recomposed and new components can be introduced at runtime. In addition, since Adaptive Java is based on standard Java, the applications composed with it can be executed with a standard Java virtual machine. Unfortunately, Adaptive Java is not an application transparent solution – rather its utilisation requires changes to the source code of a Java program.

ARCAD enables compositional adaptation to be added to Java applications in two steps [DLB01]. In the first step, generic interception hooks are woven into an existing source codes during compile time with AspectJ. Next, the intercepted operations are forwarded to the meta-level objects, which can be programmed at runtime. The meta-object code is written in Java, while the code compositional filters are written in high-level language that can be reused in programs written with other languages such as C++, if a compiler is supported for that language.

TRAP/J provides a two-step approach for dynamic adaptation of Java applications [SMCS04]. It can be used in the existing Java applications without modifying the application source code and Java Virtual Machine (JVM). TRAP/J enables developer to select during compile time a subset of classes in the existing program. TRAP/J generates aspects and reflective classes associated with the selected classes. The aspects are generic in that they simply provide hooks that intercept

the program flow. The generic MOP makes it possible to introduce new code, referred to as a delegate, to be executed upon such an interception.

Kava uses load-time byte-code rewriting to support dynamic adaptation of Java programs [WeS00]. The Byte Code Engineering Library toolkit enables Kava to incorporate its runtime meta-object protocol in a Java program. Kava allows each class to be bound to a meta-level object, where behaviours such as method invocation, method execution, and filed access can be modified dynamically. The XML-like binding language of Kava can configure the transformation process at load time. Traps inserted into class files at load time enable runtime redirection of the execution to the meta-level object.

R-Java is a Java extension supporting a type of statically-typed meta-objects called dynamic shells [Gui98]. R-Java is designed to minimize the overhead when no adaptive behaviour is required. However, the cost is a new instruction, added to the Java language that enables developers to explicitly change the class of objects at runtime. Hence, the Java virtual machine requires this modification and so R-Java is not transparent with respect to JVM.

2.3.3 Component-based adaptation

Component-based designs can support static and dynamic composition [MSKC04]. In static composition, several components can be combined during compile time to produce an application, while in dynamic composition components can be added, removed, or reconfigured within an application at runtime. Dynamic binding enables a component to call a service that another component provides. Each plugin may expose the interfaces that it provides and requires. By matching provisions to requirements, it is possible to identify components that can be connected and to create dynamic bindings between them [MDE+95].

Interposition is the addition of functionality in the midst of an existing interface boundary [SNC00]. As a result, it is possible to modify or extend programs without rebuilding them. *Adaptive methods* adapt the functional code of an application to environment changes [RGL98]. Alternative implementations with different properties can be offered for the same method. An associated selector can choose the most effective implementation at runtime [RGL98].

Design patterns [GHJV95] provide a way to reuse software designs practiced successfully for several years [MSKC04b]. For example, wrappers, proxies, and strategy pattern are used in adaptation. With wrappers objects can be sub-classed or encapsulated, so that a wrapper can control method execution. For example, the Dynamic Interposition Tools (DITOOLS) infrastructure [SNC00] uses wrappers for dynamic interposition and allows users to load and interpose new code between the call and the definition of the functions. Proxies can also surrogate objects and redirect methods calls to different object implementations. *Strategy pattern* [GHJV95] enables algorithm implementation replacements transparently.

A virtual component pattern [CSKO02] can be used in distributed applications that are executed in memory constrained embedded devices. It provides a small middleware footprint including only a minimum core and a set of virtual components, whose code can be dynamically loaded on demand [MSKC04b]. In other words, it is a placeholder inserted into the object graph and replaced as needed during program execution.

2.3.4 Middleware-centric adaptation

Instead of adapting applications, many researchers have taken a middleware-centric perspective and have investigated principles, and designed mechanisms, to achieve middleware adaptation to context [Cap03]. For example, the middleware enables the intercept and redirect of method calls and responses passed through middleware layers.

Reflective middleware may modify itself by means of inspection and/or adaptation [Cap03]. Through inspection, the internal behaviour of the system is exposed, so that it becomes straightforward to insert additional behaviour to monitor the middleware implementation. The internal behaviour of the system can be modified through adaptation.

The high abstraction level of components means that components do not reveal more details of their internal implementations than are necessary for entities in its environment to meaningfully interact with it [BCS02]. The abstraction level of components can be raised by encapsulating components to interact with their environment through well-defined interactions [BCS02]. Middleware solutions

are a common solution for encapsulation. Middleware is connectivity software encapsulating a set of services residing above the network operating system layer and below the user application layer [MSKC04b]. Middleware can be composed into four layers [Sch02] (Figure 8).

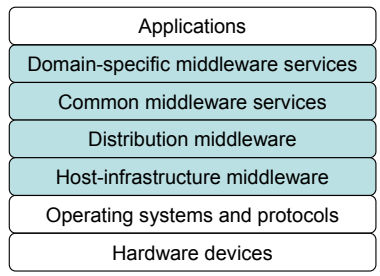


Figure 8. The layers of middleware [Sch02].

Host-infrastructure middleware resides on the top of the operating system and provides a high-level API hiding in the heterogeneity of hardware devices, operating systems, and network protocols. *Distribution middleware* provides a high-level programming abstraction, such as remote objects, enabling developers to write distributed applications in a way similar to stand-alone programs. For example, CORBA [OMG06], DCOM [Ses97], and Java Remote Method Invocation (RMI) [Dow98] all fit in this layer. *Common middleware services* contains higher-level domain-independent components that allow developers to concentrate on programming application logic, without having to write the "plumbing" code needed to develop distributed applications, instead using lower-level middleware features directly [Sch02]. Common middleware services include fault tolerance, security, persistence, and transactions involving entities such as remote objects. *Domain-specific middleware services* can offer services tailored for the particular class of applications.

The Linda model introduced tuple spaces for parallel programming [CaG89]. Tuple space exists outside of the black-box programs that do the computing. Accordingly it can be used to pass information between black boxes in different languages, between the user and system black boxes, and between the past and future black boxes [CaG89]. Linda in a Mobile Environment (LIME) middleware supports transient sharing of tuple spaces carried by each individual mobile unit [MPR01]. In addition, LIME extends Linda tuple spaces with a

notion of location and with the ability to react to the given state. Tuples On The Air (TOTA) middleware facilitates access to distributed information, navigation in complex networks, and achievement of complex coordination tasks in a fully distributed and adaptive way [MaZ04].

The middleware approach can support application transparent adaptation but is suitable only for programs that are written against a specific middleware platform [MSKC04b]. A more general approach is to implement compositional adaptation in the application program itself. The focus of this dissertation is on techniques adapting the application itself. Hence, most of the middleware solutions do not directly fall into the scope of this dissertation.

3. Dynamic component-based composition of adaptive applications

The focus of this dissertation is in adaptation that is based on dynamic composition of components. This chapter discusses techniques that support dynamic composition of component-based adaptive applications.

3.1 Introduction

Software deployment is a complex process including configuring, releasing, installing, updating, reconfiguring and even de-installing activities of a software system [HHHW97]. Context awareness plays a significant role in the component-based deployment by permitting the automatic installation and reconfiguration of a software system on the consumer site depending on the users's needs and preferences and environmental constraints [CTA+04]. This all sets requirements for components and techniques that support dynamic component-based composition.

The dynamic composition requires a notion of *plug compatibility* [NiP91, NiT95] for selecting the correct components that may be successfully combined in order to achieve a desired behaviour. Each component should have a well-defined identity that unambiguously distinguishes one component from another so that components can be accessed and manipulated specifically [BCS02].

An adaptive component-based system should support different forms and spans of life-cycles of components, including bootstrapping, deployment, installation, initialization, suspension, and termination [BCS02]. In addition, explicit construction of activities taking place in a system, the manipulation of activities, and possibly spanning or involving multiple components must also be supported [BCS02].

It must be ensured that a system executes in an acceptable, or safe manner during the adaptation process [MSKC04]. For verifying the correctness of an adapted system, developers must certify the correctness of components with respect to their specifications [BPR02]. Certification can include nonfunctional

requirements, such as security and performance, as well as functional parameters. It can be obtained by using already verified components or by generating code automatically from specifications.

Various kinds of methods are needed to support dynamic composition of components. The dynamic composition of components set requirements for:

- *Methods that are able to solve computational mismatches*. Over time, research in software engineering and programming languages has developed a number of techniques in order to overcome compositional mismatches [Sch99]. A few of these are discussed briefly in Section 3.2.
- *Architectures and frameworks supporting construction of component-based adaptive applications*. Many architectures and frameworks support dynamic composition of component-based adaptive applications. Section 3.3.
- *Techniques offering support for dynamic composition concerns of adaptive applications*. The abstraction level of component-based composition can be raised in order to make it is easier for developers to implement adaptive applications. In addition, if the composition concerns are separated from the rest of the application it may be possible to reuse composition schemas in different applications. Section 3.4.
- *Solutions supporting dynamic adaptation of distributed component-based applications.* Many solutions support adaptation of distributed component-based applications. Section 3.5.
- *Solutions supporting dynamic adaptation of content and context-sensitive applications on the client-side.* Several solutions support client-side adaptation of component-based content and context-sensitive adaptive applications. Section 3.6.
- *New adaptation techniques.* Although a great deal of solutions supporting component-based composition exist, new techniques that will make the dynamic composition of content and context-sensitive applications more fluent are still needed. Section 3.7.

3.2 Techniques for solving computational mismatches

In an ideal component world, components are available for any task that an application has to perform and these components can be simply plugged together [Sch99]. However, it is a fact that developers are often constrained to use (legacy) components that are not plug compatible with other components [ALSN01]. This can cause *compositional mismatches* [Sam97].

Control flow and *interface impediments* can make composition difficult [KMF01]. Control flow impediments relate to the ordering of execution of components [GAO95]. For example, two components cannot be used together if they have different assumptions about the sequencing of computation and passing of control between them. Interface impediments occur when components contain statically bound information about interfaces of other components, such as method names, data types, orderings, and communication protocols. However, this information will be invalid in different contexts, and will prevent the component reusing in an arbitrary composition [KMF01].

The specification of a component should also include information about its behaviour as well as its interface [Crn03]. Software components and services are not despite the common conception, Lego-like building blocks, but rather more like organs in the body, which are larger conglomerations of more minute components and provide a distinct function and goal within a context [ACM04].

This dissertation focuses on dynamic composition, which is based on components that are directly compatible or already adapted (e.g. with glue code) to work together. Techniques that solve computational mismatches are not directly in the scope of this dissertation and are therefore not discussed in detail. Only a few of them are discussed briefly in the following subsections.

Compositional mismatches can be solved with black and white-box techniques. White-box techniques focus on adapting a mismatched component by either changing or overriding its internal specification (e.g. inheritance in object-oriented languages) whereas black-box techniques only adapt interfaces [Bos99]. This dissertation considers a component as an entity which cannot be directly modified and thus considers black-box techniques only.

3.2.1 Glue-code-based solutions

Glue abstractions may bridge architectural styles and adapt components that have not been designed to work together whereas *coordination* abstractions may manage dependencies between concurrent and distributed components [AcN01]. *Glue code* can overcome compositional mismatches by adapting components to the new environment they are used in [Sch99]. It may adapt interfaces, client and server contracts, and bridge platform dependencies. Glue code may be *ad hoc*, written to adapt a single component, or it may consist of generic abstractions to bridge different component platforms [ALSN01].

Glue code can be attached to components with wrappers [GHJV95] that are an often-used technique to overcome compositional mismatches. A wrapper implies a form of encapsulation whereby a component is encased with an alternative abstraction [Sch99]. A wrapper can pack the original component into a new one with a suitable interface in order to enable the clients of the wrapped component to access the services provided by the wrapper. However, it is important to note that wrapping techniques cannot be always applied to overcome compositional mismatches [YeS97].

3.2.2 Architectural solutions

Autonomous services can avoid control model mismatches by keeping their own focus of control whereas interface impediments can be avoided by allowing services to only name their own input and output ports [KMF01]. As a result, autonomous *services* can be connected in a data flow network, where data is passed from one component's output port to another's input port according to the data flow description of the composition.

Intermediate forms focus on adapting all components of a system in a way that they conform to some standard form [Sch99]. Whereas wrappers and other glue techniques mainly focus on overcoming compositional mismatches, standard forms try to avoid them (at least to a certain degree) by restricting the kind of components which can be used in a system. Standard forms generally specify (1) how interfaces for components have to be defined, (2) what kind of data entities can be exchanged between components, (3) what kind of interaction mechanisms, and (4) what kind of architectural style(s) can be used [Sch99].

Applications based on intermediate forms tend to focus on specific application domains or architectures [Sch99].

The Blackboard architecture enables a collection of independent programs to work cooperatively on a common data structure [BMR+96]. A central control component evaluates the current state of processing and coordinates the specialized programs. This data-directed control regime is referred to as opportunistic problem solving. It makes experimentation with different algorithms possible, and allows experimentally-derived heuristics to control processing. Unfortunately, the Blackboard architecture does not support synchronization. A simple solution for synchronization problems is to save only immutable objects to the Blackboard.

Software bus and middleware solutions are commonly used intermediate form techniques. A software bus defines a standardized communication protocol for exchanging data (i.e. a set of data types that can be used to exchange data and a number of service invocation mechanisms), takes care of correct message handling, and performs all necessary data conversions. A software bus can be seen as a kind of intelligent blackboard. For example, Bart [Bea92] and Polylith [Pur94] have used the software bus mechanism. Object Request Broker middleware does not only define interface restrictions for components and interaction protocols, but also offers additional services for event models, transactions, and service traders [Sch99].

3.3 Architectures and frameworks for component-based adaptive applications

The dynamic component-based adaptation requires architectures and frameworks that help developers to implement application components and, finally, to dynamically compose adaptive component-based applications. Several adaptation techniques concentrate on the structure of component-based and dynamically adaptive applications. This section discusses a few of the most important solutions described in existing literature.

3.3.1 Lipto

Lipto is an object-oriented architecture for portable and distributed operating systems [DPH91]. It facilitates the dynamic composition of services and applications from a set of building blocks or modules.

A service class defines a downcall and an upcall interface for each of the object types which jointly provide a service in its class. A service class specifies how two layers of objects interact: it defines the downcall operations that the server objects provide and the upcall operations that the client objects must support. As a result, a set of modules which participate in the implementation of a service form a dependency graph. The module at the root presents the service to its clients whereas the modules at the leaves do not depend on any lower-level services.

The Lipto infrastructure is designed to benefit and match the needs of all parts of the system. All complex and specialized functionality is implemented in the form of subsystems consisting of a set of composable modules. For example, the location-transparent object invocation mechanism of Lipto is composed dynamically of modules. As a result, the communications services used to implement location-transparent invocations can be composed at runtime and so it is possible to take advantage of the most efficient transport protocol, depending on the location of the server object with respect to the client object.

3.3.2 Multitel

Multitel is a compositional framework for collaborative whiteboard applications supporting the coordination of users with heterogeneous multimedia and network resources [FuT99]. The Multitel platform is structured in application components and middleware platform kernel layers. Multitel composes and connects the corresponding application components dynamically and adapts them to customizable user profiles at runtime. The middleware platform kernel provides common control services for multimedia data delivery. The core of the system is the User-Service Part (USP), which represents the local service access point. It encapsulates the application architecture, which in turn drives the dynamic composition of the distributed components. The USP can also modify

its internal structure in order to add external components, such as vendor plug-and-play components.

3.3.3 LEAD++

The LEAD++ enables to the structuring of dynamically adaptive component-based software systems as a direct graph of components [AmW99]. LEAD++ introduces a software model, called DAS, and a description language for dynamic adaptation. In the DAS model, the dynamic adaptability is based on adaptive procedures. An adaptive procedure is a variant of a generic procedure (function) whose methods are selected depending on the state of its runtime environment. In addition, control mechanisms and selection strategies of adaptive procedures are realized with the user of adaptive procedures.

A LEAD++ application consists of a meta and base-level. The meta-level describes control mechanisms for dynamic adaptability. The base-level consists of primary subject domain codes in the software system. LEAD++ provides syntax to describe adaptive behaviour, which can be translated into ordinary methods in Java by the LEAD++ translator. The reflective adaptation mechanism of LEAD++ is based on dispatch objects that control adaptive procedures by using an object implementing adaptation strategies. As a result, the adaptation mechanism can be customized in a dynamic way depending on the states of runtime environments. In addition, the adaptation mechanism can be changed uniformly at the meta-level using the same mechanism as the adaptive procedures use at the base-level.

3.3.4 Fractal

The *Fractal component model* is a recursive model that allows components to be nested (i.e. to appear in the content of enclosing components) at an arbitrary level [BCS02]. A Fractal component is formed out of the controller and content parts. The content consist of (a finite number of) components, which are under the control of the controller of the enclosing component. Different components may have overlapping contents, i.e. a component may be shared by several distinct enclosing components. Fractal components can have a variable number of client and server interfaces during their life-time. A server interface can receive operation invocations whereas a client interface can emit operation

invocations. A *binding* is a connection between two or more components. The Fractal model comprises of primitive and composite bindings. A primitive binding is a directed connection between a client and server interface. A composite binding is a combination of primitive bindings and ordinary components, i.e. composite bindings are themselves Fractal components.

A component controller embodies the control behaviour associated with a particular component. It can intercept incoming and outgoing operation invocations and operation returns targeting or originating from the component's content it controls; it can provide an explicit and causally connected representation of the component's forming the content it controls; and it can superimpose a control behaviour to the behaviour of the components in its content, including suspending and resuming activities of these components. Each controller can thus be seen as implementing a particular composition operator for the components in its content.

3.3.5 One.world

The location and execution context of an application changes constantly as people move through the physical world, either carrying their own portable devices or switching between devices. *One.world* helps developers to build applications adapting automatically to an ever-changing computing environment [Gri04]. The One.world architecture provides support for contextual change, ad hoc composition, and facilitates information sharing between applications and devices by introducing four foundation services for adaptive applications. The foundation services are a *virtual machine*, *tuples*, *asynchronous events*, and *environments*.

In order to make applications usable in various kinds of devices, all code in One.world runs in a virtual machine – namely, in the Java virtual machine. One.world represents all data as tuples, which define a common data model, including a type system, for all applications and thus simplifies data sharing. Tuples are records with named and optionally typed fields. Moreover, each tuple is self-describing which enables an application to dynamically inspect its structure and contents. One.world expresses all communication through asynchronous events, which can notify applications of changes in their runtime context. Finally, like traditional operating system processes, environments host

running applications and isolate them from one another. They also serve as containers for persistent data, providing associative tuple storage and thus make it possible to group running applications with their persistent data. Furthermore, these environments nest within one another, making it easy to extend and compose applications. An outer environment has complete control over all nested environments, including the ability to easily intercept and modify events sent by inner environments to the kernel of One.world (which runs in a device's root environment) and to other devices. This interposition facility allows developers and users to dynamically change the behaviour of an application without changing the application itself. Moreover, it is particularly useful for complex and reusable behaviours, such as replicating an application's data or deciding when to migrate an application.

One.world offers system services for discovery and migration. Discovery helps in locating and connecting to services on other devices, and migration helps in implementing applications following a user through the physical world. Discovery locates resources – that is, event handlers – by their descriptions. It leverages One.world's uniform data model, in which all data, including events and queries, are tuples. Discovery uses this data model and offers, for example, early and late binding and anycast and multicast options.

Migration moves or copies an environment and all its contents to a different device, thus simplifying the implementation of applications that follow a person through the physical world. One.world moves the entire state between the devices in one atomic operation. This avoids residual dependencies and requires connectivity between the devices only during migration. As a result, many of the complexities of traditional process migration can be avoided and migration across the Internet becomes practical.

3.4 High-level programming techniques supporting dynamic composition of component-based and adaptive applications

Programming based component composition is tedious because many syntactical details that are not necessary from an architect viewpoint must be resolved [DSGO03]. Thus high-level methods are needed to facilitate the dynamic

composition of components. For example, graphical integration is easy with intuitive block diagrams as in HW/SW Codesing of an Engine Management System (VCC), but it can be difficult to manage very large designs [DSGO03]. Component integration and object interpretability can also be specified with the Uniform Modeling Language (UML) [MLG01]. Also scripts can be used in dynamic component composition.

A *script* specifies how components are plugged together [NTMS91]. Scripts can be seen as a kind of mortar "gluing" bricks (i.e. components) together [Sch99]. The essence of a scripting language is to configure components, possibly defined *outside the language*. Scripts commonly provide a higher level of programming than assembly or system programming languages, much weaker typing than system programming languages, and an interpreted development environment. Scripting languages sacrifice execution efficiency to improve the speed of development [Ous98].

A scripting language must provide (1) an encapsulation mechanism to define scripts, (2) basic composition mechanisms to connect components, and (3) abstractions to integrate components written outside the language (i.e. a *foreign code concept*) [Sch99]. A script makes architectures explicit by exposing exactly how the components are connected [ALSN01].

Domain-Specific Languages (DSLs) are programming or executable specification languages that offer, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain [DKV00]. Domain-Specific Languages (DSLs), also known as "Little Languages" [Ben86], have recently gained increasing attention [NFG06].

A *composition language* is a combination of the aspects of (1) Architecture Description Languages (ADLs), allowing us to specify and reason about component architectures, (2) scripting languages, allowing us to specify applications as configurations of components according to a given architectural style, (3) glue languages, allowing us to specify component adaptation, and (4) coordination languages, allowing us to specify coordination mechanisms and policies for concurrent and distributed components [ALSN01] (Figure 9).

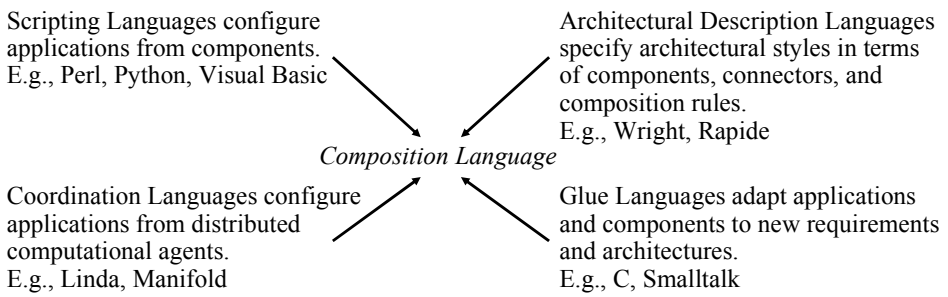


Figure 9. A conceptual framework for software composition [ALSN01].

A composition language should support the flexible construction and evolution of applications by promoting systematic component-oriented development of open systems [NiM94]. It should serve as a bridge between traditional implementation languages and higher-level composition tools. A formal object model is necessary to act as the "glue" between these layers. A composition language would function at a higher level than a programming language by allowing one to specify explicitly components, compositions, and component frameworks.

Split-level programming refers to architectural system integration and component programming on two different levels that are strongly connected by a matching class hierarchy and methods [Ous98]. Split-level programming relieves system engineers of programming artefacts and software engineering concerns specific to component implementation, and lets them focus on system architecture [DSGO03]. For example, a network simulator (NS) uses a split-programming model to create a network simulation environment with two layers of programming facilities: one for building objects and the other for composing them [BEF+00].

3.4.1 MMLite

MMLite is a modular system architecture that provides a selection of object-based components that are dynamically assembled into a full application system [HeF98]. MMLite components contain code, static data, a stack, and a number of dynamic objects. In MMLite, a component is a namespace, where components are automatically loaded on demand. When a component is loaded into an address

space it is instantiated. The instantiated component creates object instances that communicate with other objects, potentially in other components.

MMLite objects are made available to other components by registering them in a namespace. The objects expose their methods through Component Object Model (COM) [Bro95] interfaces. A proxy is interposed for delegation, if the object that is the target of a method is in a different machine or a different address space. Instead of calling the actual object, the client will call the proxy that marshals the parameters into a message and sends it for the machine where the actual object is located.

An object consists of an interface, instance pointer, implementation, and state. The interface is a list of methods. The instance pointers and interfaces are exposed to other objects; the state and the implementation are not. Worker threads execute implementation code that accesses and modifies the state. Once an object instance has been created, the instance pointer, interface, and implementation are traditionally immutable, only the state can be changed by method calls. MMLite allows runtime changes to the ordinarily immutable part of an object, even while the object is being used. For example, a method implementation of an object can be changed. A mutator thread must translate the state of the object from the representation expected by the old implementation to the one expected by the new implementation. It must also coordinate with worker threads and other mutators through suitable synchronization mechanisms. Transition functions capture the translations that are applied to the object state and to the execution state of the worker thread. Mutation enables a number of mechanisms. Interposition is done via replacement of the object with a filter object pointing to a clone of the original object. A dynamic software upgrade would replace the incorrect implementation of a method with the corrected one. Runtime code generation might use a stub implementation as a trigger. Mutation can be used to replace generic code with a specialized version that exploits partial evaluation by treating ordinarily non-constant state as immutable.

3.4.2 THINK

The THINK software framework facilitates implementation of operating system kernels and associated tools, including a library of commonly used kernel

components from fine-grained components [FSLM02]. A system built using the THINK software framework, is composed of a set of *domains* that correspond to resource, protection, and isolation boundaries. A domain comprises of a set of components interacting through bindings that connect their interfaces. In THINK, components can use bindings that are themselves assemblies of components implementing communication paths between one or more components. A binding covers both language-level bindings (e.g. associations between language symbols and memory addresses) as well as distributed system bindings (e.g. Remote Procedure Call (RPC) or transactional bindings between clients and possibly replicated servers). An interface in the THINK framework is designated by a *name*. Names are context-dependent, i.e. they are relative to a given *naming context* that encompasses a set of created names, a naming convention, and a name allocation policy.

3.4.3 OpenCOM

The OpenCOM programming model supports dynamic runtime reconfiguration of component-based systems (i.e. one can load, unload, bind, and rebind components at runtime) [CBG+04]. It can be used in a wide range of deployment environments (e.g. operating systems, Personal Digital Assistants (PDAs), embedded devices, and network processors) and allows the particularities of various deployment environments to be selectively hidden from or made visible to the programmer without inherent performance overhead.

The OpenCOM programming model has primitives that load components into units of scope and management called *capsules* and primitives that bind component interfaces and receptacles. A component can be a composite of internal sub-components and support number of *interfaces* and *receptacles* that express a dependency on an interface provided by some other component. Interface-to-receptacle *binding* is a third-party operation, i.e. code that binds a receptacle on one component to an interface on another can reside in any component within the capsule.

The OpenCOM programming model supports the notions of *caplets*, *loaders*, and *binders* as first class entities. Caplets are nested ‘sub-scopes’ within capsules; loaders provide various ways of loading components into various types of caplets; and binders provide various ways of binding interfaces and

receptacles, both within and across different caplet types and instances. Caplets, loaders, and binders are themselves implemented as components that are 'plugged-in' to hosting component frameworks.

3.4.4 BALBOA

The BALBOA component integration environment relies on smart wrappers that contain information about the types and object models of the components, and assemble them through an interpreted environment with a loose typing [DSGO03]. It is composed of three parts: a script language interpreter, compiled C++ components, and a set of split-level interfaces to link the interpreted domain to the compiled domain.

BALBOA introduces an active code generation technique, and a three-layer environment that keeps the C++ components intact for reuse. The BALBOA composition environment has an architecture definition, intermediate wrapper, and component definition layers. At the architecture definition layer, the type of a component is abstracted and a type management system is used to infer and instantiate the exact types required by the simulation model. Designs are assembled from configured components by using the Component Integration Language (CIL). The component definition layer is the bottom layer, which can contain any compiled C++ object. The intermediate wrapper layer is the link between the interpreted and compiled layer.

CIL interpreter can execute a number of commands to perform component composition, simulation control, test bench creation, and event monitoring. A type-inference system maps all weakly-typed CIL interfaces to strongly typed C++ component implementations to produce an executable architectural model. CIL can generate Split-Level Interface (SLI) wrappers around each C++ component and component libraries. SLIs can be used to select, adapt, and validate the implementation types.

3.4.5 LuaCorba

LuaCorba facilitates the development, rapid prototyping, and testing of distributed auto-adaptive applications by offering high-level programming mechanisms to access the dynamic interfaces provided by CORBA [MUCR02].

It allows developers to define adaptation strategies for distributed applications that can dynamically select the components that best suit their requirements, verify whether the system satisfies these requirements, and eventually react to variations in the non-functional properties of the used services.

The adaptation strategies are encapsulated to smart proxies. A smart proxy represents a service and dynamically selects the specific server that will actually provide the service. The programmer can define in the smart proxy the behaviour that best suits the application requirements. Examples of such behaviours are: component substitution according to execution conditions, choice of different components for different requested operations, use of alternative methods, and management of control mechanisms.

To verify and guarantee the fulfilment of the requirements over time, the smart proxy can observe and control the properties associated with these requirements. The LuaMonitor mechanism informs the smart proxies about relevant changes in the observed properties, and allows them to activate, when appropriate, the programmed adaptation strategies.

The adaptation strategies are defined with an interpreted Lua language. As a result, the reconfiguration facilities are transparent to the functional behaviour of applications, i.e., the reconfiguration code is not mixed with the application code. In addition, the reconfiguration solution can be updated dynamically and applied in various applications and with components with different functional interfaces.

3.4.6 CASA

CASA provides a runtime system that supports adaptation at various levels of an application – from lower-level services to application code [MuG05]. The CASA runtime system monitors the execution environment on behalf of the running applications and carries out the adaptation of the affected applications. The adaptation policy of every application is defined in a so-called application contract, which is specified by using an XML-based language facilitating modification, extension, and customization of the adaptation policy at runtime. Thus, the adaptation concerns are separated from the business concerns of an

application. In addition, the user or administrator has control over the adaptation policy, although the adaptation is carried out in a user-transparent manner.

3.4.7 Component Configurators

The Component Configurator [KoC00, Kon00] defines a framework for reconfiguring distributed applications [SEK03]. It simplifies requirement and architectural analysis by defining a clear separation between the aspects related to environment monitoring, detection of environmental changes, and dynamic application reconfiguration. The framework is composed of a collection of CORBA objects and services. The framework is implemented with Java and uses JacORB as the CORBA ORB.

The Component Configurators coordinate reconfiguration actions between the application components. Each application component has a corresponding Component Configurator that keeps track of the dynamic dependencies between the component and other system or application components. By maintaining an explicit representation of those dependencies, it is possible to guarantee runtime consistency.

Component Configurators disseminate events across inter-dependent components. Examples of common events are the failure or migration of a component, internal reconfiguration, or replacement of a component implementation [SEK03]. Those events affect all the dependent components and coordinate reconfiguration actions between the application components.

Programmers can insert the code to a Component Configurator to deal with configuration-related events. The framework organizes the code that treats each environment event as a set of strategies, using the Strategy Pattern [GHJV95]. This provides a clear separation of concerns between the application functional code and the code supporting application reconfiguration [SEK03].

3.4.8 Plasma

The *Plasma* framework [LaH05b] is based on the Fractal component model [BCS02] and supports building of self-adaptive component-based applications and hierarchical composition of components. It provides a dynamic Architecture

Description Language (ADL) that enables developers to describe the dynamic behaviour of an application with respect to changes in the environment. The middleware framework offers the required tools to match a specification with a component assembly. Adaptive behaviour is executed by using probes, sensors, and actuators. Probes and actuators sense the context whereas actuators perform the actual adaptation actions. For example, functional, structural, and policy reconfigurations can be made.

3.5 Solutions supporting dynamic adaptation of distributed component-based applications

Dynamic adaptation of a complex distributed software system requires considerable formal knowledge about the specifications of the system, ways to express, reason about, and coordinate the adaptation of system components, and computational mechanisms for interfacing to system components and modifying their functionality on the fly [VKK01]. Many solutions support adaptation of distributed component-based applications.

3.5.1 Sparkle architecture

The Sparkle architecture provides an Internet-enabled infrastructure for pervasive computing and dynamic component composition [BWL03]. It consists of client devices, facet servers, intelligent proxies, and execution surrogates. Client devices provide a platform for the application to run whereas a facet server is a place for the service providers to publish and release their designed facets. Intelligent proxies receive requests from the client and return suitable facets. Execution surrogates are resource rich machines responsible for carrying out executions for the clients just in case the client devices do not have enough resources.

A facet consists of the shadow and code segment parts. Shadow describes the properties of the facet in XML format. For example, it can define the identifier, vendor, and version of the facet. It can also define an identifier for the functionality of the facet, resource requirements, and dependencies to other facets. Code segment is the body of the executable code implementing the

functionality. Facet dependencies are the functionalities that a particular facet depends on.

A facet does not interact with the user and does not maintain any application state. Data and execution state are stored in the container that provides an application-like abstraction to the user. Sparkle makes the adaptation mechanisms transparent for the programmer. A facet programmer needs to provide different versions of facets and application programmers need to specify functionalities making up an application. The adaptation policies are carried out by the resource manager and the proxy server of Sparkle.

3.5.2 WebCODS

The WebCODS system facilitates dynamic component composition over the Web [SPW02]. It supports dynamic component loading and unloading to providers, clients, and the component broker, transferability of WebCODS components, instantiation and execution of components, and dynamic composition with other components.

WebCODS provides a set of connectors enabling precompiled Java components to be linked together. In addition, these connectors can be rewritten using reflection methods. Existing Java application components are converted into WebCODS components with wrapper classes that characterize the interface of the component, provide interaction points for connectors to communicate with the component, and finally specify the execution sequence of the component. Connectors link components dynamically by using the reflection methods of Java. Adaptations may be needed when different components are involved. For example, customization of the parameters of connectors and the generation of component-specific glue-code to handle the connections can be needed.

3.5.3 Hadas

A Hadas programming model enables distributed applications to be adapted dynamically to changing circumstances [BHL01]. It enables distributed applications to be composed dynamically of independently developed and autonomously maintained components, so that components can be changed independently with no impact on the application [BHL01]. With it a deployed

component can be tailored to the target application and environment without necessarily knowing in advance the complete details of the environment. In addition, with Hadas, deployment can be optimized based on fresh runtime information [BHL01].

Hadas provides a set of programming and runtime tools, including a component builder, a distributed component browser, and a graphical shell for Hadas-based application execution [BHL01]. Hadas components are fully reflective and support both dynamic introspection and evolution. Each component is split into fixed and extensible sections. The fixed section is treated as conventional class-based items that cannot be changed during the lifetime of the component. This portion of the component can be used to store its fundamental state and behaviour. The extensible aspect of Hadas comprises the mutable portion of the component though which the structure and behaviour of the component can be changed with meta-methods, and in which new items (data, objects, or methods) can be added, removed, or changed on-the-fly. Since the items of the extensible section cannot be counted to have specific semantics at any given time, they cannot be reused in other instances.

The Hadas universe is structured as a collection of sites, each containing a collection of components [BHL01]. The hierarchy defines a naming scheme for components since they are uniquely identified by their site's address, name, and the path from the site to them. Sites are linked to each other with the site-level Link protocol, which negotiates and establishes a mutually agreeable connection and a channel of communication. The dynamic application is centred on the notion of *Ambassador*, which is a representative of a component that gets deployed to remote site upon successful completion of a connectivity protocol. Ambassador serves as an adaptive remote reference and an interoperability handler.

In Hadas, an Ambassador is a representative of a component that gets deployed to a remote site upon successful completion of a connectivity protocol. Ambassadors serve as an adaptive remote reference and as an interoperability handler that can transform the input from an invoking component in the Ambassador's deployed site into a format that is expected by the Ambassador's home component. It then transforms the output from the home component into a format that is expected by the remote invoking component.

3.5.4 AMPROS

The AMPROS (Adaptive Middleware Platform for PROactive reconfigurable Systems) architecture provides a distributed, just-in-time, and context-aware deployment of component-based applications [ATB04]. The AMPROS middleware platform aims to hide as much as possible the details of the hardware, the operating system, and the telecommunication protocols from application developers and users [CTA+04]. A context manager is associated with a middleware manager coping with the collaboration between the users and the other entities of the middleware. The users and middleware services rely on context information: the former for expressing needs and behaviour, the latter for being proactive. These middleware services include disconnection management, fault management, and deployment services.

AMPROS allows the installation and reconfiguration of an application according to the context by using a set of lightweight adaptive components that are able to modify several deployment parameters such as the architecture of the application and the placement and configuration of the components. In addition, the adaptive components may be added to existing deployment tools without modifying the tools themselves [ATB04].

3.5.5 Kinesthetics eXtreme

Kinesthetics eXtreme (KX) is a mobile agent-based infrastructure for runtime monitoring and reconfiguration of component-based distributed systems [VKK01]. The KX meta-architecture aims to handle global situations, perhaps involving heterogeneous components obtained from multiple sources where it would be difficult if not impossible to retrofit self-assurance. The occurrence of significant conditions within the target system are detected and reported by the monitoring part of the meta-architecture. The process engine is notified about conditions and may dynamically instantiate, initialize, and finally dispatch one or more software agents, called *Worklets*, to perform the adaptation operations.

Each Worklet can contain one or more mobile code snippets, called *worklet junctions*, to actuate the required adaptation of the target system. The data structures of a junction can be initialized with data, typically coming from the task definition, process context, and information contained in the event(s) that

represents the triggering condition. Furthermore, any process-related configuration of Worklets is accounted for *worklet jackets,* which allow scripting of certain aspects of Worklet behaviour in the course of its route.

The process engine requests junctions for the dynamic adaptation task at hand from a Worklet Factory, which has access to a categorized semantic catalogue of junction classes and instantiates them on its behalf. Interfaces exposed by junctions in the catalogue must be matched to the kind of capabilities that are necessary for the task and to descriptions of the target components subject to dynamic adaptation. Once a Worklet gets to a target component, the interaction between the junction(s) and the target component is mediated by a *host adaptor*, which semantically resolves any impedance mismatch between the interface of a junction and that of the component.

3.5.6 SOCAM

A service-oriented context-aware middleware (SOCAM) is an infrastructure for building and prototyping context-aware applications in a smart-home environment [GPZ04]. It is built on top of the service-oriented OSGi [OSGi03] that is an open specification for a component model allowing networked services to be deployed and managed. Adding an OSGi Service Platform to a networked device adds the capability to manage the life cycle of the software components in the device from anywhere in the network. Software components can be installed, updated, or removed on the fly without having to disrupt the operation of the device.

SOCAM components are independent Java-based service components that can be distributed over various networks. Components can interact with each other by using Java RMI that lets distributed objects invoke each other's methods. SOCAM offers a set of services for context discovery, acquisition, and interpretation and provides an ontology-based context model that leverages the Semantic Web technology and Web Ontology Language (OWL). OWL is an ontology mark-up language that enables context sharing and reasoning.

SOCAM provides methods that are able to reason about various contexts and derive high-level contexts from low-level ones and implicit contexts from explicit ones. Thus the acquired contexts are converted into a semantic space

where context-aware applications can share and access them. In SOCAM, *context providers* abstract acquired contexts and convert them to OWL representations so that other service components can share and reuse them. The *context interpreter* provides logic reasoning services to process context information. The *service-locating service* enables the users and applications to locate context providers and interpreters. Application developers can predefine rules and specify the methods to be invoked when a condition becomes true. Rules are saved in a file and preloaded into *context reasoners*. These rules can also be later updated dynamically.

3.5.7 Other solutions for component-based deployment

Several component-based deployment solutions such as the CORBA Component Model component packaging and deployment model [OMG02], the EJB [Sun02] and the .Net deployment solutions [Sco00], and the J2EE deployment API [Sea02] exist. However, these deployment solutions do not consider the context information.

A distributed system can be composed of a collection of interacting Web services. A web service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described, and discovered by XML artefacts and supports direct interactions with other software applications using XML-based messages via Internet-based protocols [ABFG04]. A Service-Oriented Architecture (SOA) defines the services of which the system is composed, describes the interactions that occur among the services to realize certain behaviour, and maps the services into one or more implementations in specific technologies [BJK03]. In other words, it is a way of designing a software system to provide services to either end-user applications or other services through published and discoverable interfaces [BJK03]. Web services use often Simple Object Access Protocol (SOAP) [BEK+00] for interaction. The Web Services Description Language (WSDL) [CCMW01] supports the description of Web services whereas the Universal Discovery, Description, and Integration (UDDI) standard [CHRR04] is developed to support the discovery of Web services.

The OMG has specified a data model for a deployment plans that can contain information about artefacts that are part of the deployment, how to create

component instances from artefacts, where to instantiate them, and information about connections between them [OMG03]. The specification also presents a data model for describing the domain into which applications may be deployed as a set of interconnected nodes with bridges routing between interconnects.

3.6 Client-side solutions for adaptive content and context-sensitive applications

Content adaptation has been the subject of much research. However, content adaptation techniques (e.g. [MSL99, KAK+00, LaH05]) concentrate more on server-side adaptation. This dissertation focuses on client-side adaptation and on composition of adaptive content and context-sensitive applications of software components. Thus the content adaptation techniques do not directly fall into the scope of this dissertation and are not discussed in more detail.

Mobile browsers are developed in a software product line for reducing costs, adding reusing, and improving quality [Jaa02]. An Extensible Browser Architecture (EBA) [SCH+04] that can be extended with widgets is also proposed. In addition, an adaptive browser [HeI01] that is based on the Grail (*http://grail.sourceforge.net/*) browser and is able to adapt for display type, bandwidth availability, and network connections is developed, too. CM4DG is an XML-based visual component model for designing dynamic GUIs [XPJ03]. It states relations among various visual components and composition rules. Multi-User Publishing Environment (MUPE) (*http://www.mupe.net/*) is an open source platform for creating mobile multi-user context-aware applications. The application functionality and downloadable UI descriptions are programmed into the MUPE server. As a result, the client implemented with Java MIDP is the same for the different MUPE applications. Users can modify the virtual world of MUPE with specific built-in tools. For example, end-users can create and publish data into the system. The drawback of MUPE is the amount of transferred data between the client and server.

Approaches are also proposed to support client-side processing in mobile browsers. For example, mobile browsers can execute programs, called dynamic documents [KPT94], and generate the actual information that is displayed to the user. Dynamic documents can define actions that may access information local

to the client, fetch other documents, and finally generate an HTML document to be displayed for the user.

Asynchronous JavaScript and XML (Ajax) [Gar05] is a standard-based programming technique designed to make Web-based applications more responsive, interactive, and customizable [Smi06]. Ajax is based on asynchronous JavaScripts that can be embedded in an HTML page to make server calls, retrieve new data, and simultaneously update the Web page without having to reload all the contents, all while the user continues interacting with the program [Pau05]. The Document Object Model (DOM) enables the interactive user experience. Furthermore, XML and XSLT are used in data exchange and transformations. JavaScript joins the components together whereas asynchronous communication between the client and server is done via *XMLHttpRequests*.

In addition to content browsers, adaptive client applications are developed to be used in pervasive environments, too. Solutions that facilitate the construction of adaptive applications of smart spaces are also proposed. For example, dynamically generated workflows are used to coordinate services in pervasive computing environments and to allow customized user-environment interaction [RaM04]. Both context information and user preferences can be utilized in workflow generation.

Rule-based languages are used in the definition of adaptive behaviour. For example, the ECA rule specification language [ZhB04] that defines Event, Condition, and Action parts for each adaptation rule is used in adaptive applications. Furthermore, the proposed visual tools [ZhB04] enable end-users with limited coding skils to program adaptation to better correspond to their needs.

A *universal interactor* approach allows the user to select objects from the list of services and to control these objects with UIs presented in the client device [HKSR97]. The semantics of the control interfaces of services are described with an Interface Definition Language (IDL). The clients can first fetch the IDL files for services. Then, the entire service Graphical User Interface (GUI) is transferred to the client if a language implementation for a GUI that the client can display is available. Finally, the GUI is augmented with an interface description that starts with base data types and allows them to be executed

dynamically. If a GUI is not available, a rough GUI is generated for the control functions of a discovered object interface. The clients can prefetch the IDL files for active services. Delays can be further minimised through mobility prediction [LiM96], allowing pre-fetching in response to assumptions about user mobility patterns. ICrafter [PLF+01] facilitates the creation of UIs for combinations of services. In it, appropriate generic or customized UI generators are selected to compose a UI for context and for appliance and service descriptions.

3.7 Task-based composition technique for adaptive content and context-sensitive applications

Sections 3.3, 3.4, 3.5, and 3.6 discuss the methods supporting dynamic composition of component-based adaptive applications. An overview of the characteristics of these methods is presented in Figure 10.

Many infrastructures and solutions support sensing the context and dynamic composition of component-based adaptive applications. For example, (e.g. Lipto, Multitel, AMPROS, Component Configurators, Fractal, and One.world) platforms and frameworks support dynamic composition of component-based and adaptive context-aware applications. In addition, languages are defined for the composition concerns of adaptive applications. For example, BALBOA and LuaCorba offer languages enabling developers to define adaptation strategies for component-based adaptive applications.

Although, a lot of solutions supporting dynamic component-based composition exist, new techniques that will make the dynamic composition of content and context-sensitive applications more fluent are still needed. For example, in a comparison of ubiquitous Web application approaches [KPRS03] it was noted that in most approaches adaptation is machine controlled. In addition, it was noted that adaptation is done primarily from scratch. As a result, existing approaches do not take advantage of the performance gains which could be achieved when re-using already adapted application elements [KPRS03]. For example, it should be possible to reuse adapted UI parts (e.g. XML-based UI descriptions and icons) when the application is adapted for new contexts. This can reduce the amount of network traffic and processing related to adaptation and thus improve the performance of adaptation.

Class	Name	Description	Adaptation Type	Reactive Adaptation	Proactive Adaptation	Speculative Adaptation	Separation of data and functionality	Scripting Language for Adaptation Concerns	Active Context-Awareness	Passive Context-Awareness	Fault Tolerance	Context-Sensitive Handling for Failures	Caching of Context-Sensitive Application Instances	Caching of Adaptation Tasks
Architectures and frameworks for component-based and context-aware adaptive applications	Lipto [DPH91]	An object-oriented architecture for portable and distributed operating systems	C	✓	(✓)				✓	(✓)	(✓)			
	Multitel [FuT99]	A compositional framework for collaborative whiteboard applications	C	✓					✓					
	LEAD++ [AmW99]	Structures an adaptive software system as a direct graph of components	R	✓				✓	✓	(✓)				
	Fractal [BCS02]	A recursive model for nested components	C	(✓)			(✓)							
	One.world [Gri04]	Architecture for adaptive applications using shared data	C	✓			✓		✓		✓			
High-level programming techniques supporting dynamic composition of adaptive applications	MMLite [HeF98]	Architecture for object-based components	C	✓					✓					
	THINK [FSLM02]	Framework for OS kernels and associated tools	R	(✓)					(✓)					
	OpenCOM [CBG+04]	A component model for adaptive software systems	R	✓					✓					
	Balboa [DSGO03]	An integration environment for components	R	(✓)				✓						
	LuaCorba [MUCR02]	CORBA-based technique for distributed auto-adaptive applications	C	✓	✓			✓	✓	(✓)	(✓)			
	CASA [MuG05]	A runtime system for adaptive applications	C	✓				✓	✓					
	Component Configurators [KoC00, Kon00]	A framework for distributed adaptive applications	R	✓	✓				✓		✓			
	Plasma [LaH05B]	A framework for self-adaptive applications	C	✓	✓		(✓)	✓	✓					
Solutions supporting dynamic adaptation of distributed component-based applications	Sparkle architecture [BWL03]	Infrastructure for pervasive computing and dynamic component composition	C	✓			✓	✓	✓					
	WebCODS [SPW02]	Supports dynamic component composition over the Web	C	(✓)				✓	(✓)					
	Hadas [BHL01]	A programming model for adaptive distributed applications	C	✓	✓			✓	✓	✓				
	AMPROS [ATB04]	A platform for proactive reconfigurable systems	C	✓	✓				✓	✓	✓	✓		
	Kinesthetics eXtreme (KX) [VKK01]	A mobile agent-based infrastructure for distributed and adaptive software systems	C	✓	✓		✓	✓	✓	✓				
	SOCAM [GPC04]	An OSGi-based component model for deploying and managing networked services	C	✓				✓	✓					
	Service-Oriented Architecture (SOA) [BJK03]	Architecture for composition of interacting Web services	C	✓				✓	✓					
	OMG's Deployment Plans [OMG03]	A data model for deployment plans	R	✓			✓	✓	✓	✓	✓			
Client-side solutions for adaptive content and context-sensitive mobile applications	An Extensible Browser Architecture (EBA) [SCH+04]	Architecture for browsers that can be extended with widgets	C	(✓)										
	Adaptive Web Browser [Hel01]	An adaptive browser that provides context-aware behaviours and UIs	C	✓	✓				✓	✓	✓			
	CM4DG [XJP03]	An XML-based visual component model for designing dynamic GUIs	C	✓				✓	✓					
	MUPE [http://www.mupe.net/]	A Multi-User Publishing Environment (MUPE) application platform	C	✓				✓	✓					
	Dynamic Document [KPT94]	A method that is able to generate HTML documents to be displayed for the user	C	✓	✓	(✓)		✓	✓	✓	(✓)		(✓)	
	Ajax [Gar05]	Asynchronous JavaScript and XML	C	✓	✓	(✓)		✓	✓	✓	(✓)			
	Dynamically generated workflows [RaM04]	Models a user's interaction with the environment based on workflows	C	✓				✓	✓	✓	✓	✓		
	ECA [ZhB04]	A three-tier approach for context-aware applications	C	✓				✓	✓					
	Universal Interactor [HKSR97]	Allows the user to select and to control services with UIs presented in the client device	C	✓	✓	(✓)		✓	✓	✓				
	ICrafter [PLF+01]	Facilitates the creation of UIs for combinations of services	C	✓				✓	✓					
	TaskCAD [Pal05, Pal07]	A composition technique for content and context-sensitive applications.	C	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

C = Context-aware adaptation **R = Resource-aware adaptation** ✓ ■ **Supported** **(✓) = Partially Supported**

Figure 10. Characteristics of the methods supporting dynamic composition of component-based adaptive applications.

The mobile environment poses challenging requirements for adaptation techniques. The mobile usage is spontaneous and thus it is important that applications are adapted as fast as possible for rapidly changing contexts. For example, a technique supporting speculative adaptation and both caching of context-sensitive asynchronous adaptation tasks and application instances prepared with tasks is needed. At the same time it should be possible to use various kinds of adaptation components in adaptation.

More precisely, it should be possible to utilise various kinds of adaptation actions that can modify and insert new context information to the adaptation requests, compose application instances for them, and also adapt the prepared application instances. Various kinds of adaptation actions may require different kinds of execution structures. For example, sequential and concurrent executions are needed for context-sensitive adaptation actions. Actions that enable the end-user to control the adaptation process are also needed. In addition, context-sensitive methods are required to handle the errors that are raised while an application is adapted for a new context. It is not, however, possible to provide ready-made executions for all kinds of adaptation actions. Thus methods that enable developers to implement new kinds of executions for adaptation actions and to utilise them in various kinds of adaptive applications are needed, too.

In order to solve the described problems, this dissertation introduces a task-based composition technique for component-based and adaptive content and context-sensitive applications. Particularly, as will be shown in this dissertation, the task-based composition technique helps developers of adaptive applications in the following tasks:

- The technique helps developers to define adaptation strategies where applications are composed in many phases and where both application instances and asynchronous adaptation tasks are cached. As a result, it enables developers to implement speculative adaptations for applications. In addition, the technique separates adaptation concerns clearly from the business logic of an application. As a result, adaptation strategies can be defined separately, changed at runtime, and may be possibly reused in various applications.

- The technique enables developers to utilise various kinds of context-sensitive actions in adaptation and to configure them to work in different application environments.
- The technique offers special executions for adaptation actions, enabling the end-user of an adaptive application to control the adaptation process. In addition, it offers structures that can define context-sensitive handling for errors raised while an application is composed. Furthermore, developers can extend the task-based composition technique with new implementations that can introduce execution structures for the defined adaptation actions. For example, new conditional executions that can utilise data available in the application environment and control execution of the defined adaptation actions can be implemented and utilised in task-based adaptation.

4. Solution: A task-based composition technique for adaptive content and context-sensitive applications

This chapter defines the refined requirements for the task-based composition technique and gives a general insight of the technique. It is organized as follows. A brief introduction is provided in Section 4.1. Task-based adaptation using factories is discussed in Section 4.2. Then, Section 4.3 discusses how the most suitable adaptation elements can be selected for context. An environment for fine-grained and reusable adaptation actions is introduced in Section 4.4. Caching of context-sensitive tasks and application instances is discussed in Section 4.5. A language for specifying task-based adaptive applications is introduced in Section 4.6. Task-based speculative adaptation is discussed in Section 4.7. Then, the utilisation of the task-based composition technique is discussed in Section 4.8. Finally, Section 4.9 gives few usage scenarios for the task-based composition technique.

4.1 Introduction

Object and component-oriented technologies are the state of the art in general purpose software technologies [RBH04]. Unfortunately, they lack, several assets that are required fulfil the criteria of the ubiquitous computing paradigm: they are not easily adaptable and not resource aware [RBH04].

The mobile environment poses many challenging requirements for dynamic composition techniques (see Chapter 1). A proper technique supporting the dynamic composition of context-aware applications (see Chapter 2) makes the composition process more effective and easier. Although, a lot of solutions supporting the dynamic component-based composition exist (see Chapter 3), new techniques that will make the dynamic composition of content and context-sensitive applications more fluent are still needed.

The task-based composition technique supports dynamic composition of adaptive component-based applications. In the sequel, we will use the term *"task-based"* to denote our approach for adaptive component-based composition.

Design patterns [GHJV95] are generic solutions that are commonly utilised in many applications. In general, a name, problem, solution, and consequences are the essential elements of a pattern notation [GHJV95]. The problem describes when to apply the pattern; the solution describes the elements that make up the design, their relationships, responsibilities, and collaborations. Finally, the consequences are the results and trade-offs of applying the pattern. This dissertation uses a same kind of notation that is used in design patterns and describes problem, solution, and summary of the solution for each part of the task-based composition technique. In addition, an example is provided for each part. The example solutions are later utilised in case studies. However, it must be noted that although the task-based composition technique is a generic solution that is utilised in different kinds of adaptive applications (see Chapter 7) it is not a commonly utilised design pattern.

4.2 Task-based adaptation using factories

4.2.1 Problem

The nature of the heterogeneous mobile environment sets many requirements for adaptive applications. For that reason, it can be difficult to compose adaptive content and context-sensitive applications dynamically. Mobile usage is spontaneous and requires efficient adaptation mechanisms. A technique that can perform concurrent reactive and proactive adaptations and compose an application in multiple phases for various contents and contexts is needed. In addition, it must be possible to change the adaptation elements dynamically to better fit to different kinds of usage environments and contexts.

4.2.2 Solution

We used the content adaptation model of W3C (see Section 1.1.1) as a base and developed the *task-based composition technique* [Pal05, Pal07] to facilitate dynamic composition of component-based applications that are adapted for various kinds of contents and contexts (Figure 11).

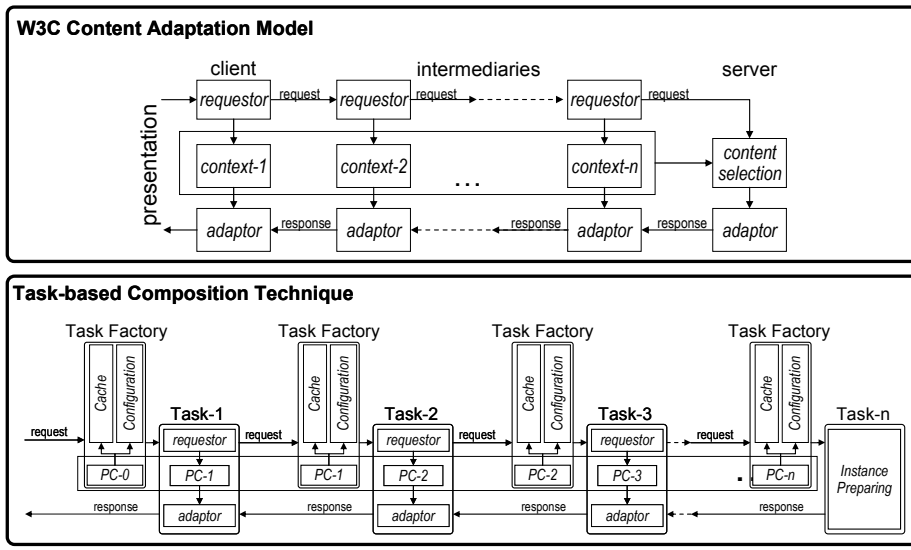


Figure 11. The task-based composition technique is based on the content adaptation model of W3C.

Like the requestor-adaptor elements of the W3C model, a *task* is an adaptation element that can provide additional context information, request other tasks, adapt their responses, and finally deliver responses for the requestors. The context can change constantly and so a snapshot of the context, called *Processing Context* (PC) is needed to ensure that the context values used in the adaptation are immutable. For example, part of the delivery context [Lew03] attributes can be added to the PC. A *request* defines an *identifier*, *data sources*, and *PC* for a task that composes application parts for requests and delivers them in responses. A task can deliver multiple responses during its execution. For example, it can deliver an error response for the requestors and then continue execution. Tasks are executed simultaneously. The synchronous requestors can wait for the response of the task or then wait until the task is completely executed. In addition, the response of the task can be refreshed and delivered for the requestors after the task is executed.

A task can be in an *out of action*, *pending*, *suspended*, *ready*, or *failed* state (Figure 12). Before a task is executed, it is in an out of action state. During execution, the task is in the pending state. Task execution can be suspended, too. It will wait in the suspended state until it is activated to resume its execution.

After execution, the task is set to the ready state. If errors are raised during execution, the task is set to the failed state. Finally, in order to improve performance, it is possible to reset a task instance that is not suitable for the current or any potential future PC. It can be later reused and started to process a new request.

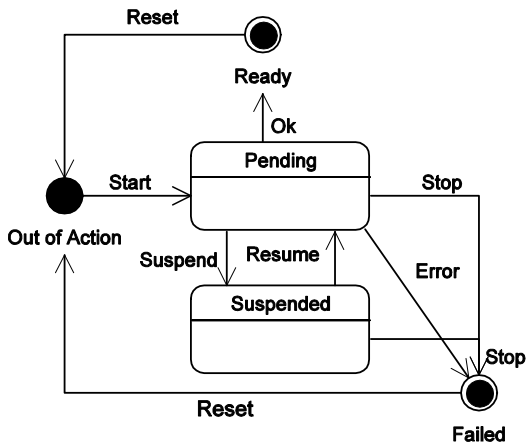


Figure 12. The states of a task.

All requested tasks cannot be necessarily successfully executed. For example, if a mobile device is disconnected, it is not possible to fetch contents from the Web. In order to prevent deadlocks, a time-out can be defined for task requests. As a result, if a task does not respond during the defined time-out, an error response is passed for the requestor.

It must be possible to configure the adaptation elements for different kinds of applications and contexts. Thus the task-based composition technique extends the content adaptation model of W3C with *task factories* that can construct various kinds of tasks for requests and specific processing contexts.

The task-based composition technique can be used in both *reactive* and *proactive* adaptation [Cap03] (Figure 13). A requestor can observe the context and request reactive adaptation tasks to adapt the application for new PCs, if the specific parts of the context are changed. Proactive adaptation tasks deliver refreshed responses composed for the processing context.

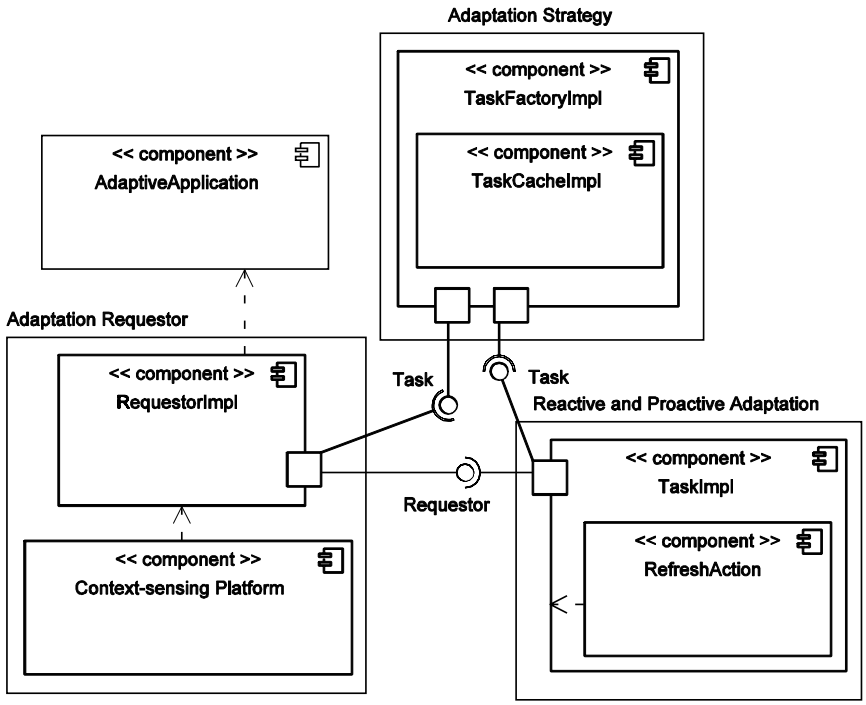


Figure 13. Tasks support reactive and proactive adaptation.

4.2.3 Summary of the solution

A *task* is an adaptation element that is able to provide additional context information, request other tasks, adapt their responses, and finally deliver the following kinds of responds for the requestors:

- A *task ready response* is delivered after the task is executed.
- An *update response* is delivered when the response of the task is refreshed.
- An *error response* is delivered if errors are raised during task execution.
- A *task failed response* is delivered if task execution is failed.

Concurrent tasks can do both reactive and proactive adaptations and compose applications in multiple phases for new processing contexts. In addition, adaptation strategies can be changed dynamically by replacing the task factories with new ones.

4.2.4 Example

An adaptive application can be divided into model, view, and controller parts according to the *Model-View-Controller (MVC)* pattern, first introduced in the Smalltalk-80 system [KrP88]. The application platform can observe the context and request tasks to compose models, views, and controllers for the new context, if the specific parts of the context are changed. Tasks compose these parts for the request and PC and finally deliver them in responses (Figure 14).

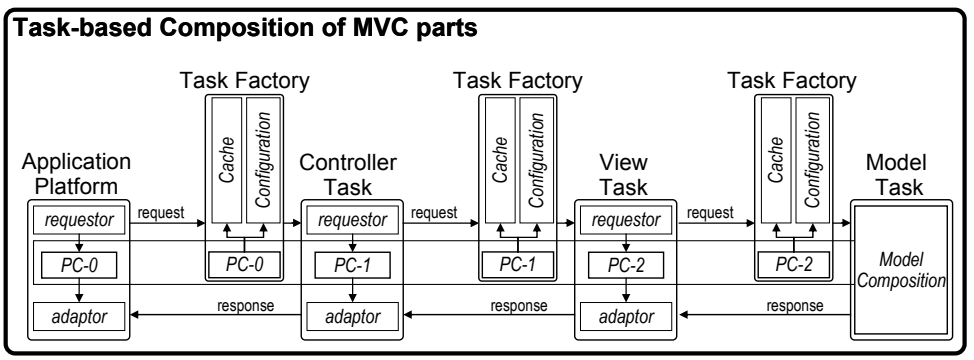


Figure 14. Tasks can compose the model, controller, and view parts of an adaptive application.

Tasks can also refresh MVC parts and deliver them for the requestors. For example, the model task can perform proactive adaptation and deliver a refreshed model for the task that composes views for the model and pass them for the controller task that will attach controllers to the composed views. Finally, the model-view-controller parts are delivered for the application platform that will finally display the refreshed MVC parts for the user.

4.3 Selecting the most suitable context-sensitive elements for adaptive applications

4.3.1 Problem

Various kinds of *Context-Sensitive Elements* (CSEs) (e.g. tasks and application instances) are used in dynamic composition of adaptive applications. This

requires methods that are capable of selecting the correct CSEs for various PCs. It must be possible to define how suitable a CSE is for different PCs, to calculate suitability values for alternative CSEs, and, finally, to select the most suitable CSEs for PC.

4.3.2 Solution

A numerical *Suitability For Context (SFC)* value defines how suitable a CSE is for a specific PC. SFC value is zero for elements that are suitable for all PCs, positive for CSEs that are suitable for given PC, and negative for CSEs that are not suitable for PC. Thus, if alternative CSEs are available, the CSE that has the highest positive SFC value is selected.

Context elements define suitable PC values for a particular CSE. For example, they can define (e.g. *Boolean*, *literal*, *numeric*, *time*, and *location)* attributes, logical (e.g. AND, OR, XOR, and NOT) expressions, and references to other context elements. In addition, new context element types can be added if new kinds of PC values are needed in adaptation. The context elements are built into a tree structure to form an overall presentation for suitable PC values. A *weight factor* (default value is 1) defines the importance of a context element. The context elements use the defined weight factors, calculate SFC values, and together form the overall SFC value for the CSE and PC. The following rules are used when SFC values are calculated for the context elements:

- *Context attribute (e.g. Boolean*, *literal*, *numeric*, *time,* and *location) elements*. The SFC value is the defined weight factor if PC is suitable for the context attribute. Otherwise the SFC value is the defined weight factor multiplied with -1.
- *Logical AND element*. The SFC value is the sum of the weight factor of the AND element and the SFC values of the child context elements if all the child context elements are suitable for PC. Otherwise, the SFC value is the weight factor of AND element multiplied with -1.
- *Logical OR element*. The SFC value is the sum of the weight factor of the OR element and the maximum of the SFC values of the child context elements if one or more child context elements are suitable for PC.

Otherwise, the SFC value is the weight factor of OR element multiplied with -1.

- *Logical XOR element*. The SFC value is the sum of the weight factor of the XOR element and the maximum of the SFC values of the child context elements if exactly one child context element is suitable for PC. Otherwise, the SFC value is the weight factor of XOR element multiplied with -1.
- *Logical NOT element*. The SFC value is the suitability value of the child context element multiplied with -1, if the child context element is not suitable for PC. Otherwise, the SFC value is the weight factor of the NOT element multiplied with -1.
- *INCLUDE element*. The SFC value is the SFC value of the included context element.

A task knows the processing context in which it is executed. Thus it is possible to compare given and cached PCs. A simple solution is to accept only processing contexts identical with PC of the cached task. However, this may cause processing overhead if only a small number of PC attributes affect task execution. Context elements are capable of identifying equivalent processing contexts. Thus the utilisation of cached tasks can be increased, if the PC values that affect task execution are defined precisely for the task. Now, according to context elements, PC1 and PC2 are equivalent:

1. If the same attributes are defined in PC1 and PC2 and
2. PC1 and PC2 give equivalent suitability for context (SFC) values for the context elements that are related to the task.

However, it is not always possible to define beforehand the PC values that will affect task execution. For example, the task can request other tasks asynchronously and the user can control task execution. As a result, the context dependencies of a task can change at runtime. For example, an XHTML page may have several references to other contents, which can be downloaded asynchronously with context-sensitive tasks. In addition, tasks can add information (e.g. time attribute) to PC before requesting other tasks. As a result, although related context definitions are known, it is not possible to compare the given PC to the PC of the task correctly because the given PC may not have all

the context attributes that affect execution of the task. A simple way to avoid this problem is to prevent the caching of that kind of task. Unfortunately, this will cause overhead because tasks may be constructed and started in vain. However, it must be recognized that a task started for a new PC may possibly utilize part of the cached application instances that the previously started tasks have prepared. In addition, context-sensitive tasks can be divided into smaller subtasks in order to improve performance. As a result, part of these subtasks may not be context-sensitive and so it may be possible to utilise part of their responses in different contexts.

4.3.3 Summary of the solution

The context elements are built into a tree structure that describes suitable PC values for a particular context-sensitive element (CSE). As a result, it is possible to define suitable PC values for alternative CSEs, calculate SFC values for them, and to select the most suitable CSEs for PC when an adaptive application is composed. Context elements are also able to identify equivalent processing contexts and can thus improve the utilisation of cached tasks.

4.3.4 Example

For example, the end-user can use an adaptive application at home in the evening. Thus it must be possible to select the correct adaptation elements and to adapt the application for this PC. The following kinds of context elements can define suitable PC values for the "*Home in the evening*" context (Figure 15).

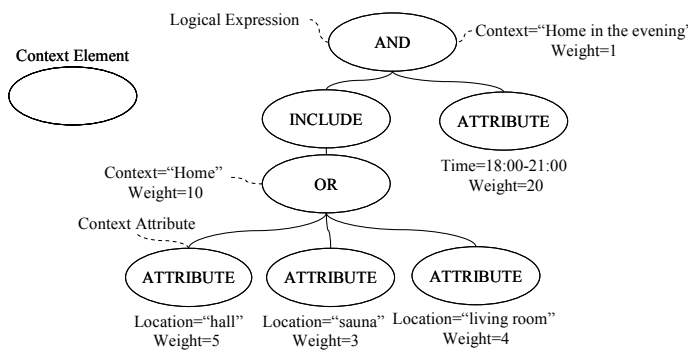


Figure 15. Context elements that define that a CSE is suitable for a "Home in the evening" context.

For example, there can be the following kinds of PCs:

PC1={location=”living room” and time=10:00}
PC2={location=”living room” and time=19:00}
PC3={location=”living room” and time=19:30}
PC4={location=”sauna” and time=20:00}

As a result, the SFC value is for PC1, PC2, PC3, and PC4 is:

SFC(PC1)=-1
SFC(PC2)=1+10+4+20=35
SFC(PC3)=1+10+4+20=35
SFC(PC4)=1+10+3+20=34

Context elements are able to identify equivalent PCs:

ISEQUIVALENT(PC1,PC2)=false
ISEQUIVALENT(PC2,PC3)=true
ISEQUIVALENT(PC2,PC4)=false

PC2 and PC3 are equivalent because all context elements give same SFC values for PC2 and PC3.

4.4 An environment for fine-grained and reusable adaptation actions

4.4.1 Problem

Making a request, starting a thread for a task, and finally composing a response for the request all cause processing overhead. Multiple adaptations can be done in a single task and thus it is possible to reduce processing overhead related to execution of tasks. However, it is difficult to reuse extensive tasks in various applications.

It must be possible to compose tasks of smaller adaptation components and to reuse these in various kinds of tasks. This sets requirements for methods that enable these adaptation components to compose content and context-sensitive application instances in different environments.

4.4.2 Solution

A task can be composed of smaller context-sensitive adaptation elements, called *actions* that can fetch application instances from various sources and compose new ones to different targets. Thus actions can utilize existing application instances and possibly make composites of them. Adaptation actions can use the reflection [Mae87] methods for dynamic component-based composition. For example, these methods enable actions to load application components and to construct instances for them at runtime. Construction of component instances can be simplified by composing an application of components offering empty constructors and named methods for configuring the constructed instances. For example, this approach is used in Java Beans [DeK05].

A task request is handled in the following phases:

1. *A task factory constructs a task for the request and PC.*
2. *The task is then started and it will call its actions that are suitable for PC to process the request.* An action name and event is passed for the called action that executes the requested processing and returns a result or throws an exception if its execution fails. The action event offers access to the task factory and thus enables the action to request other tasks, too.
3. *Finally, after the actions are executed, the response of the task is delivered for the requestors.*

It must be possible to reuse action implementations in various applications. An *Instance Reference* (*IRef*) is an identifier for an application instance and its environment. It can also define URI for the content that is related to the instance. *Input and output mappings* support reuse of adaptation actions and enable an *environment adaptor* to replace IRefs used by an action with IRefs used in the application and thus configure an action to fetch application instances from certain sources and to add them to specific targets.

Input mappings can also configure an action to utilise context-sensitive *settings* that define alternative configuration parameters for various PCs. The most suitable parameters are selected for PC and passed for adaptation action. Thus it is possible to adapt the behaviour of actions for various contexts.

Many existing solutions support hierarchical composition. For example, Fractal [BCS02] and One.world [Gri04] introduce architectures for application component composites and offer support for making these application composites to co-operate. Similarly, as is the case in One.world, application instances can be composed to nested data storages, called *environments* (Figure 16).

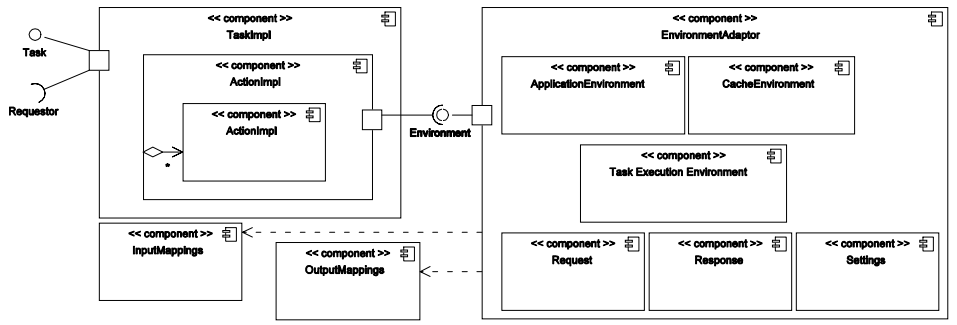


Figure 16. The actions of a task can compose application instances to various environments.

For example, the application environment contains instances that are used actively in the application. Temporary instances, that are needed only when the task is executed, are saved to the task execution environment. Instances that may be needed in the future are added to the cache environment.

4.4.3 Summary of the solution

A task is composed of smaller context-sensitive adaptation elements, called *actions* that can utilize existing application instances and possibly compose composites of them. Context-sensitive settings configure an action to work in various kinds of application environments. For example, the input and output mappings configure actions to fetch application instances from certain sources and to add them to specific targets.

Actions can utilise the application instances of the task execution, cache, and application environments and the application instances defined in the request and add application instances to the response of the task. As a result, the actions can easily utilise previously constructed instances and so do not prepare new

ones in vain. In addition, input mappings can also configure an action to utilise context-sensitive parameters in adaptation.

4.4.4 Example

For example, context-sensitive settings can be defined for an action that composes visualisation elements (glyphs, see Section 4.9.1) for content elements and adds the presentation (*glyphframe*) to the response of the task (Figure 17).

The input elements enable the action to fetch *frameName* and *frameURI* from the request and a *glyphDefinition* that describes visualization elements and parameters for them (e.g. font, color, and position parameters) from the settings (Figure 18). Finally, the output element defines a target IRef that makes the action to compose the *glyphFrame* instance to the response of the task.

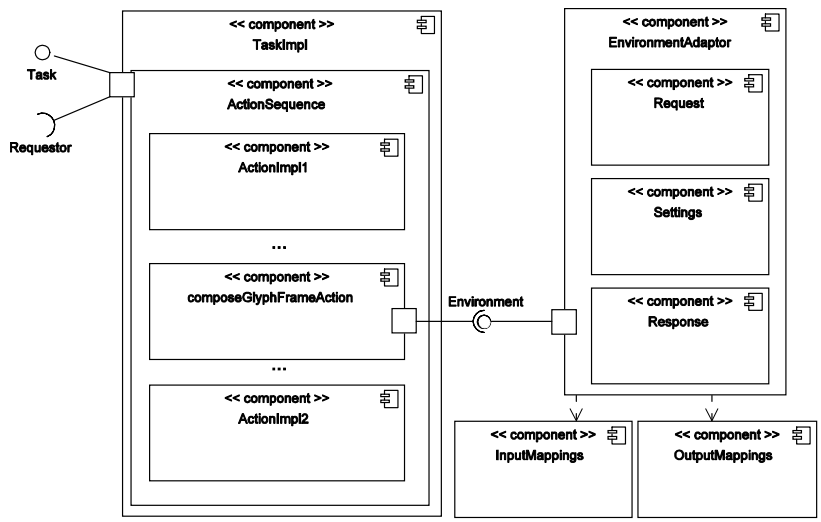


Figure 17. An action that composes visualization elements for content elements.

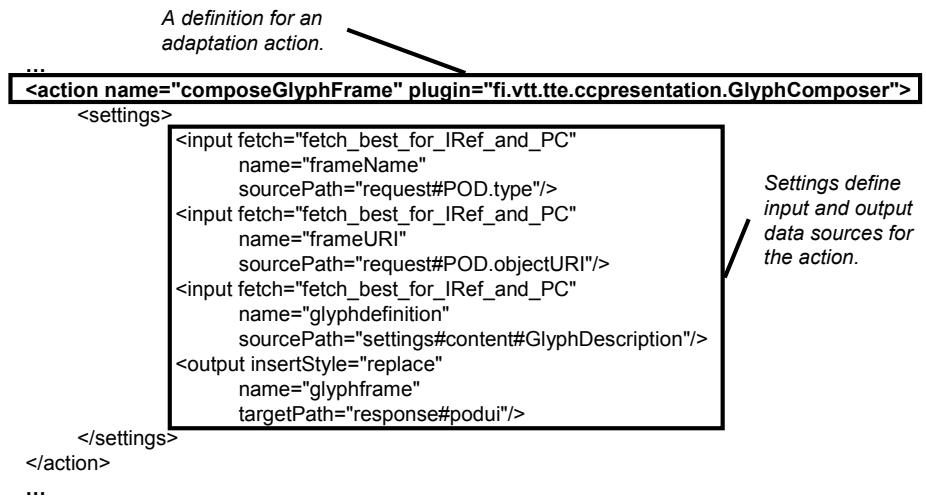


Figure 18. A definition for an action that composes visualization elements for content elements.

4.5 Caching of context-sensitive tasks and application instances

4.5.1 Problem

Dynamic composition can offer applications that fit better to the context. However, it can be difficult to compose adaptive applications dynamically for possibly rapidly changing contexts. This sets requirements for methods that facilitate:

Tasks caching. Many parts of the application can request the same adaptation tasks. Thus the caching of tasks can speed up the adaptation process. As a result, new tasks are not started in vain but rather previously started tasks are utilised in adaptation. However, it is not trivial to manage the cache of context-sensitive tasks which can have dependencies on other tasks, too. The challenge is to identify when a task is suitable for a specific request and PC. Part of the cached tasks may be expired, too. This all sets requirements for methods that can control caching of context-sensitive tasks, select the most suitable tasks for PC, and finally enable the application to utilise them in adaptation.

Application instance caching. The caching of application instances enable adaptation tasks to utilise the already prepared application instances and thus speed up adaptation. Different application instances may need various kinds of caching schemas. For example, some application instances may be only valid at a certain time. In addition, in order to enable fetching of context-sensitive instances from the cache, it must be possible to define which kinds of contexts an application instance is suitable for. Thus methods are needed to control caching of context-sensitive applications instances and to enable applications to fetch the most suitable application instances from the cache.

Reducing the number of cached tasks and application instances. In order to minimize the usage of the limited memory of a mobile device, it must be possible to reduce the number of cached tasks and application instances. For example, the tasks and application instances that are not suitable for current or potential future PCs must be removed from the cache.

4.5.2 Solution

The task factory constructs a new task only if a task suitable for the request and PC is not found from the cache. Tasks are searched in three phases from the cache:

1. *The tasks having a correct name are searched from the cache.*
2. *The tasks that are suitable for the request are selected from the found tasks.* A task is suitable for the given request only if its request and the given request have identical parameters.
3. *The most suitable task is selected for PC.* If a cached task is not found, a task configured for the identifier and PC is constructed and added to the cache.

Separate *RequestAcceptors* implementations can introduce specialised caching schemas for tasks and will thus improve caching. They can utilise the context elements that are defined for the task and decide for what kind of requests and PCs a specific task is suitable.

Similarly, *context comparators* can control the caching of context-sensitive application instances, utilize the context elements, and define how suitable an

application instance is for various PCs. For example, an application instance is removed from the cache if a context comparator defines that the instance is expired. Output mappings can define context comparators and thus introduce specialised caching schemas for the application instances added to the defined targets.

A maximum size can be defined for the task and application instance caches to reduce the usage of the limited memory of a mobile device. The number of cached tasks and application instances can be reduced in the following steps:

1. If the size of the caches is too big, tasks and application instances that are not suitable for the current or possible forthcoming PCs can removed from the caches first.
2. If the size of the caches is still too big, the tasks and application instances that are not suitable for the current or the most probable possible forthcoming PCs can be removed from the caches, too. If more memory is still needed, the rest of cached tasks and application instances that are not suitable for the current PC can also be removed.
3. Finally, if the size of the caches is still too big, the rest of cached tasks and application instances can be removed, too.

The cache can be refreshed at different points. For example, one solution is to refresh the cache always when new tasks are requested or when the current PC or possible forthcoming PCs change. In addition, the application can request cache refresh if there is not enough free memory available.

4.5.3 Summary of the solution

Different kinds of application instances and tasks may need various kinds of caching schemas. *RequestAcceptors* and *ContextComparators* can introduce specialised caching schemas for context-sensitive tasks and applications instances and thus improve their caching.

4.5.4 Example

For example, the implementation for the task-based composition technique (see Chapter 6) offers *RequestAcceptors* that are able to control caching of:

- *Not context-sensitive tasks.* This *RequestAcceptor* checks only that the given request equals the request of the task. As a result, the task is suitable for all PCs.
- *Context-sensitive tasks utilising only part of the PC values*. This *RequestAcceptor* works as the previous one but it will check only those attributes of PC that affect execution of the task. The composition schema can define the named context elements related to the task. As a result, the RequestAcceptor will accept the request if the given PC and PC of the task are equivalent.
- *Tasks that are suitable for specific PC only*. This *RequestAcceptor* checks that the given request equals the request of the task. In addition, it will check that given PC and PC of the task are identical.

In addition to these, developers can improve the usage of the cached tasks and decrease the memory consumption with new RequestAcceptor implementations that are able to identify more precisely for which kinds of requests and PCs a certain task is suited and to request a task to be removed from the cache, when the response of the task has expired.

The implementation for the task-based composition technique (see Chapter 6) offers ready-made *ContextComparators* for:

- *Not context-sensitive instances*. This context comparator is used for application instances that are suitable for all contexts.
- *Instances that are suitable for defined PC values only*. This context comparator uses the named context elements and calculates the suitability value for given PC. The suitability value is the sum of suitability values calculated for named context elements and PC. The context-sensitive instance that has the highest suitability for context (SFC) value is selected.
- *Instances that are suitable for specific PC only*. This context comparator compares given PC to PC of the instance. The instance is suitable only if PC and PC of the instance are identical.

In addition to these, developers can improve the usage of the cached application instances and decrease the memory consumption with new ContextComparator implementations that are able to identify more precisely for which kinds PCs a certain application instance is suitable or when the instance has expired and must be removed from the cache.

4.6 A language for specifying task-based adaptive applications

4.6.1 Problem

It must be easy for developers to utilise composition tasks in various kinds of applications. In addition, new requirements may emerge for an adaptive application. Thus it must be possible to change both the components and adaptation policies of an adaptive application so that the application will better correspond to the new requirements.

Rather than defining composition tasks inside the application logic, an interpretable Domain Specific Language (DSL) [DKV00] can be defined to offer ready-made structures and notations for task-based composition. As a result, the adaptation concerns are clearly separated from the business logic of the application. In addition, it is possible to change the adaptation policies at runtime.

Adaptive applications set many requirements for the task-based composition language. Firstly, it must offer basic notations for adaptation tasks and actions. Secondly, it must be possible to describe context-sensitive settings for reusable adaptation actions that will configure those actions to work in different execution environments and processing contexts. This requires notations that enable developers to describe the acceptable PC values for adaptation actions, settings elements, and application instances that actions prepare.

4.6.2 Solution

The *task-based composition language* enables developers to define *composition schemas* that describe tasks and context-sensitive actions and settings for composition of adaptive applications. Figure 19 presents a meta-model that

defines the structure and basic elements for the language that enable developers to use various kinds of components in task-based composition.

Some of the most important elements and attributes of the language are discussed in the following paragraphs. In addition, the language can be extended with new action, context, and content elements.

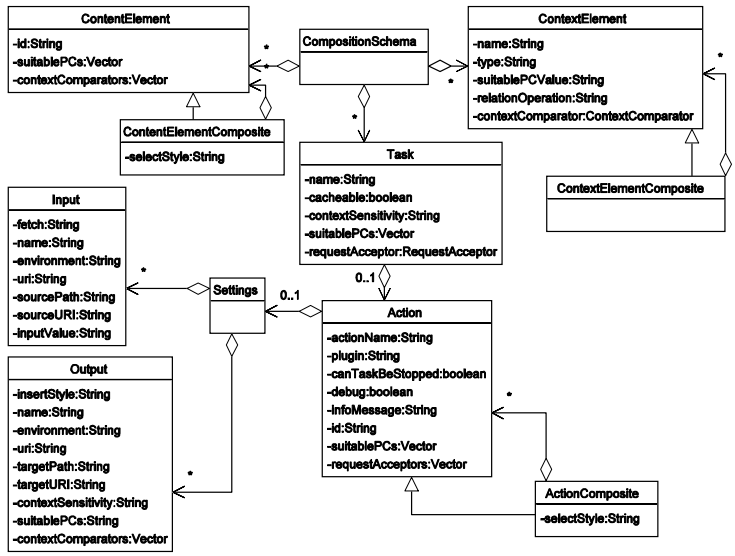


Figure 19. A meta-model for task-based composition language.

Task element

The *Task* element defines a name and caching attributes for a task. The task is saved to the cache if the *cacheable* attribute has a value "yes". If the *contextSensitivity* attribute has a value:

- "*full*", the task is suitable if the given PC and the PC of the task are identical.

- "*partial*", the task is suitable only if given PC and PC of the task are equivalent (see Section 4.3). The *suitablePCs* attribute defines the context elements describing the PC values that affect task execution.
- "*none*", the task is suitable for all PCs.

In addition, the *requestAcceptor* attribute defines plugins to control caching of the task.

Action elements

An *Action* element enables developers to use various kinds of action plugins in tasks. It has *actionName*, *plugin*, *canTaskBeStoppedBefore*, *suitablePCs*, *requestAcceptors, debug,* and *id* attributes. Plugin defines a class path for the action plugin. The actionName attribute value is passed for the plugin when it is called to execute a certain kind of action. The task can be stopped before the action is executed if the *canTaskStoppedBefore* attribute has a value "yes". The action can provide debug information for the developer if the debug attribute has a value "yes". The *infoMessages* attribute enables developers to define messages to inform the end-user about the progress of the adaptation. The defined messages are passed for the users of the task factory before the action is executed. In addition, the messages can also be delivered in the responses for the requestors of the task. The *id* attribute defines a unique identifier for an action. Thus a requestor of a task can refer to the action and define settings for it. The *suitablePCs* attribute defines the named context elements that calculate a SFC value for given PC and for the action. The action is executed only, if it is suitable for given PC and the request acceptors of the action accept the request.

Various applications require different kinds of executions for adaptation actions. However, it must be noted that it is not possible to provide ready-made executions for all kinds of adaptation actions. Thus the language offers the *ActionComposite* element that is able to introduce new kinds of executions for adaptation actions. Developers can use the element and define a plugin that implements a new execution structure for the actions of the action composite.

Input and Output elements

The *Input* and *Output* elements define mappings for the *Instance References* (IRefs) used by an action and thus enable it to fetch application instances from

different sources and to add them to various targets. Input elements can also define textual parameter values or to refer to context-sensitive *ContentElements* and thus configure the behaviour of the action.

The *name*, *environment*, and *URI* attributes of the Input and Output elements specify for which IRefs the defined mappings are used. An Input element defines *sourcePath*, *sourceURI*, and *fetchstyle* attributes for an application instance that an action tries to fetch. The *sourcePath* attribute defines where the instance is fetched. It is formatted as follows:

Variable = $[sourcePath]$
InstanceName=#[Identifier][Variable]
ParameterName=.[Identifier][Variable]
DataRef = [InstanceName]ParameterName][DataRef]
EnvironmentName = [Identifier][Variable]/[EnvironmentName]
sourcePath = [EnvironmentName][DataRef][Variable].

The collection of all instances that are found for IRef is returned if the fetchstyle attribute has a value "*fetch_all_for_IRef*". The collection of instances that are found for the IRef and are suitable for PC is returned if the fetchstyle attribute has a value "*fetch_all_for_IRef_and_PC*". Finally, only the most suitable instance for IRef and PC is returned if the fetchstyle attribute has a value "*fetch_best_for_IRef_and_PC*".

The Output element defines mappings for IRef when an action composes an application instance to a specific target. If the *insertStyle* attribute has a value "*replace_existing_instances*", the instances stored for the IRef are removed before the new one is added. The *targetPath* attribute defines the place where the instance is composed. It is formatted as follows:

targetPath = [EnvironmentName][DataRef][Variable].

The *contextSensitivity*, *suitablePCs*, and *contextComparators* attributes of the output element can define various kinds of caching schemas for the application instances that are composed to various targets. The *suitablePCs* attribute defines the named context elements that calculate a SFC value for given PC. If the *contextSensitivity* attribute has a value:

- "*full*", the application instance is suitable if given PC and PC of the application instance are identical.
- "*partial*", the named context elements or context comparators define how suitable the application instance is for PC.
- "*none*", the application instance is suitable for all PCs.

Context elements

The *ContextElement* elements describe acceptable PC values for context-sensitive elements that are utilised in the adaptation. A context element describes a suitable PC value and a context comparator for it. It can also define a relational operation attribute that has a value (e.g. equal, unequal, lower than, lower than or equal, greater than, greater than or equal) that defines how the defined PC value is compared to the specific value of PC. The *ContextElementComposite* element defines a context comparator that is capable of calculating an SFC value for PC and for the context elements that it contains.

Content elements

The *ContentElement* element defines context-sensitive contents for adaptation actions. The *suitablePCs* attribute defines the named context elements that calculate a SFC value for given PC. In addition, the *contextComparators* attribute can define the plugins that are able to calculate a SFC value for PC and for the content element.

The *ContentElementComposite* element can contain alternative content elements that are configured for various PCs. As a result, it is possible to select the most suitable content elements for various PCs. The *selectStyle* attribute defines how the elements are selected. More precisely, if the *selectStyle* attribute has a value:

- "*all*", all the content elements that are suitable for PC are selected.
- "*first",* the first suitable content element for PC is selected only.
- *"best",* the most suitable content element for PC is selected.

4.6.3 Summary of the solution

The task-based composition language makes it easier for developers to use the task-based composition technique in various kinds of applications. Developers can use the language and define *composition schemas* that describe adaptation tasks and context-sensitive actions and settings for them and configure task factories to compose adaptation tasks for various kinds of component-based adaptive applications. As a result, it is possible to implement application-transparent adaptations in which the adaptation logic of an application is developed separately and clearly extracted from the application logic. In addition, it is possible to download new composition schemas at runtime and to change the adaptation strategies dynamically.

4.6.4 Example

We have used the meta-model as the basis for a language to specify task-based composition schemas. Figure 20 depicts a refined meta-model for this language.

A few of the most important elements and attributes of the refined meta-model of the language are introduced in the following paragraphs.

Context elements

Logical *And, Or, Xor,* and *Not* and *Include* context element composites and *Boolean, Number, NumberRange, Literal, Time, TimeRange, Date, DateRange,* and *Location* context attribute elements enable developers to define suitable PC values for context-sensitive adaptation elements. The *Include* element is able to attach other context elements to a context element composite.

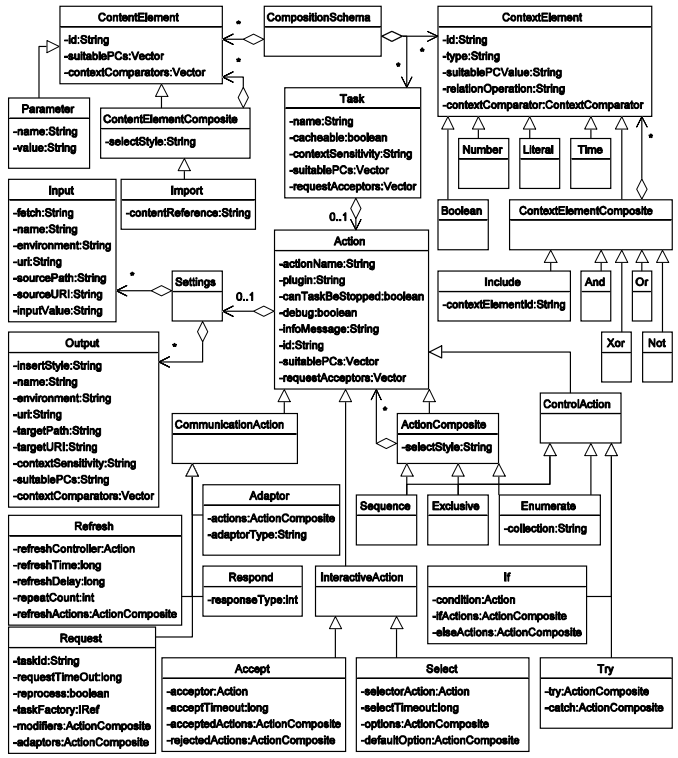


Figure 20. A refined meta-model for the task-based composition language.

Content elements

ContentElements define context-sensitive parameters and contents for actions. The *Parameter* element defines a name and value for a context-sensitive parameter. An *Import* element can include other content elements to the content definition. Developers can also implement context comparators to calculate suitability for context (SFC) values for new kind of PC values and use them in composition schemas.

Execution structures for actions

In order to facilitate the task-based composition of adaptive applications, the task-based composition language offers communication, control, and interactive structures for actions. These are discussed in the following paragraphs.

Structures for communication actions

Task-based composition requires *communication structures* that enable developers to define:

- *Requests for tasks and adaptors for the received responses.* The language must offer a structure that enables developers to define tasks that are able to request other tasks, modify requests, and adapt the received responses. In addition, in order to prevent deadlocks, it must be possible to define time-outs for task requests. As a result, an exception is thrown if a response is not delivered during the defined time-out. Furthermore, developers do not necessarily know how the application should be composed for various contexts in the development time. Thus the language must offer structures that make it possible to update adaptation strategies at runtime.
- *Respond to requestors*. A task can deliver multiple responses during its execution. Actions that can pass unfinished responses for the requestors or respond and stop task execution are needed. Actions that can pass error responses for the requestors are needed, too.
- *Refresh the response of the task*. Part of the adaptation tasks can require post-processing. For example, the response of a task that downloads information from the Web may expire. The language must offer structures that enable developers to define post-processing actions for the tasks.

Thus the language offers communication elements that enable tasks to request other tasks (*Request* element), to deliver responses for the requestors (*Response* element), and to refresh the response of the task (*Refresh* element).

The Request element defines a name for the requested task and a time-out for the request. By default, the request is passed for the task factory that has composed the task of the request action. In addition, the composition schema can have actions that first download composition schemas and then initialise new task factories for them. In this case, the request action is configured to pass a request for the initialized task factory. If a *reprocess* attribute is set to a request, a new

task is constructed and started although a cached task is available for the request and PC. At the same time, the old task is replaced with the new one.

By default, the input attributes of the request of the task are included in the new request. The request can contain actions that will modify the request before it is passed for the task factory and adaptor actions that are later notified about the responses that the requested task delivers. The request action can contain various kinds of adaptor action composites that will handle all the responses or only certain kinds of responses. The *adaptorType* attribute defines which kinds of responses will be passed for the actions of the adaptor action composite. More precisely, if the *adaptorType* attribute has a value:

- "*all*", all the responses will be passed for the actions.
- "*ready*", only task ready responses will be passed for the actions.
- "*update*", only the task update responses will be passed for the actions.
- "*failed*", only the task failed responses will be passed for the actions.

The adaptors can handle the responds and possibly modify the response of the owner task. As a result, the requestors of the owner task are also notified about the updated response.

The *respond* element defines an action that delivers the response of the task for the requestors. The *refresh* element defines adaptation actions to do post-processing related to the task. The *refreshTime*, *refreshDelay*, and *repeatCount* attributes define when, how often, and how many times the defined refresh actions are executed. In addition, the *refreshController* attribute can define a plugin that sets the refresh attributes at runtime.

Control structures for actions

Different applications require different kinds of executions for adaptation actions. To be usable, the language must offer ready-made *control structures* for:

- *Sequential execution.* In order to enable a task to execute multiple adaptation actions, action composites that can execute action sequences are needed.

- *Exclusive execution.* It must be possible to configure alternative actions for various contexts and to execute them exclusively so that only the most suitable action for the request and PC is executed.
- *Context-sensitive error handling.* Errors can be raised while an application is adapted for a new context. The language must enable developers to define context-sensitive handling for errors raised while adaptation actions are executed.

The *Sequence* element can contain multiple actions that are executed one after the other. The *Exclusive* element can contain multiple actions. However, if the *selectStyle* attribute has a value "*first*", only the first suitable action for PC is executed. If the *selectStyle* attribute has a value "*best*", the most suitable action for PC is executed.

The *If* element can define a condition, multiple if and else-actions, and a plugin that decides whether the if or else-actions are executed.

The *Try* element enables developers to define context-sensitive adaptation actions for exceptions. The thrown exception is passed for the first catch action composite that is suitable for PC.

The *Enumerate* element offers a control structure that repeats the defined actions for all the child objects of a named data object.

Structures for interactive actions

The dynamic composition may take a long time (e.g. more than ten seconds), if a part of the resources used in composition are located in the Web. Thus it is important to give feedback for the user about the progress of adaptation. In addition, it must be possible for the user to control and to select between alternative adaptation actions. To be usable, the task-based composition language must offer ready-made structures for *interactive actions* that provide:

- *Feedback messages.* The user must be informed about the progress of the adaptation. Thus the language must offer structures enabling developers to define context-sensitive feedback messages to the actions of the tasks. As a result, it is possible to display these messages for the user when he or she is using the adaptive application.

- *User controls.* The language must offer action structures that enable the user to control the adaptation process. For example, it must be possible for the user to accept or select between alternative adaptation actions. In order to decrease the coding effort, UIs and info messages can be described with common mark-up languages (e.g. XHTML) and presented in a specific way for the user of an adaptive application.

The *Accept* element enables developers to define a structure that has an acceptor action that can display user interfaces and thus enables the user to control the adaptation. After the user has made a selection, either the accepted or rejected actions are executed. The user may not be concentrated on using the application forever. In this case, a time-out can be defined for accept. As a result, the accept actions are executed if the user does not reject the actions during the defined time-out.

The *Select* element can define a structure that enables the end-users to control adaptation. The selector actions can present a user interface enabling the user to select between alternative adaptation actions. After the user has selected, the actions of the selected option are executed. The default actions are executed if the user does not select any option during the defined time-out.

Summary of the elements of meta-models

We have used the meta-model as the basis for a language to specify task-based composition schemas. It defines the core elements for the task-based composition (Figure 21). The refined meta-model introduces new elements for the task-based composition (Figure 22). For example, it offers new structures for communication, control, and interactive actions and elements for content and context descriptions.

	Element	Attribute	Description	Type
	Composition Schema		Describes task, content, and context elements.	Element
Task Description	**Task**		Defines a task and adaptation actions for it.	Element
		name	Defines a name for the task.	Literal
		cacheable	Controls task caching. The task is cached if the attribute has a value “yes”.	yes | no
		contextSensitivity	Defines the context-sensitivity of the task.	full | partial | none
		[suitablePCs]	Defines the context elements that affect task execution.	Context1, Context2,…
		[requestAcceptors]	Defines request acceptors to control the task caching.	Plugin Description List
	Action		Defines an action that is used in task-based composition.	Element
		actionName	Defines a name for the action.	Literal
		plugin	Defines a plugin implementation for the action.	PluginDescription
		canTaskBeStopped	If the attribute has a value "yes", the task can be stopped before the action is executed.	yes | no
		[id]	Defines a unique identifier for an action	Literal
		[debug]	If the attribute has a value "yes", the action can provide debug information.	yes | no
		[suitablePCs]	Defines the context elements that calculate a SFC value for the action and PC.	Context1, Context2,…
		[requestAcceptors]	The action is executed only, if the defined request acceptors accept the request.	Plugin Description List
		[infoMessage]	Defines a message that informs the end-user about the progress of the adaptation.	Literal
	Action Composite		Defines a new execution structure for the child actions.	Element
	Input		Defines an IRef mapping that enables the action to fetch application instances from different sources.	Element
		fetch	Defines how the applications instance is fetched for IRef.	fetch_all_for_Iref | fetch_all_for_IRef_and_PC | fetch_best_for_IRef_and_PC
		name	Defines a name for the needed application instance.	Literal
		[environment]	Defines an environment for the needed application instance.	Literal
		[uri]	Defines URI for the content that relates to the application instance.	URI
		sourcePath	Defines a source for the application instance.	IRef
		[sourceURI]	Defines URI for the content that relates to the source application instance.	URI
		[inputValue]	Defines a textual input value for the action.	Literal
	Output		Defines an IRef mapping that enable the action to add application instances to various targets.	Element
		insertStyle	Defines how the application instance is added to the defined target.	Add | Replace_Existing_Instances
		name	Defines a name for the application instance.	Literal
		[environment]	Defines an environment for the application instance.	Literal
		[uri]	Defines URI for the content that relates to the application instance.	URI
		targetPath	Defines the target for the application instance.	IRef
		[targetURI]	Defines URI for the content that relates to the target application instance.	URI
		[contextSensitivity]	Defines the context-sensitivity of the application instance.	full | partial | none
		[suitablePCs]	Defines the context elements that calculate a SFC value for PC.	Context1, Context2,…
		[contextComparators]	Defines context comparators that calculate how suitable the instance is for PC.	Plugin Description List
Content Elements	**Content Element**		Define context-sensitive content for actions.	Element
		id	Defines a name for the content element.	Literal
		[suitablePCs]	Defines the context elements that calculate a SFC value for PC.	Context1, Context2,…
		[contextComparators]	Defines context comparators that calculate how suitable this element is for PC.	Plugin Description List
	ContentElement Composite		Defines a composite of context-sensitive content elements.	Element
		selectStyle	Defines how the elements are selected for PCs.	all | first | best
Context Elements	**ContextElement**		Describes a suitable PC value and a context comparator for it.	Element
		id	Defines a name for the context element.	Literal
		type	Defines a type for the context element.	Literal
		suitablePCValue	Describes what kind of PC values are suitable for this context element.	Literal
		[relationOperation]	Defines how the suitablePCValue is compared to the given PC value.	e.g. equal, unequal, lower than, lower than or equal, greater than, greater than or equal.
		[contextComparator]	Defines context comparator plugin to calculate a SFC value for a specific PC value and for the context element.	Plugin Description
	ContextElement Composite		Defines context elements and a context comparator to calculate a SFC value for PC and for the defined context elements.	Element

[AttributeName] = optional attribute

Figure 21. The core elements of the task-based composition language.

	Element	Attribute	Description	Type
Content Elements	**Parameter**		Defines a context-sensitive parameter.	Element
		name	Defines a name for the parameter.	Literal
		value	Defines a value for the parameter.	Literal
	Import		Include other content elements to the content definition.	Element
		contentReference	Defines a content source that is imported to the content definition.	Literal
Context Elements	**Boolean**		Defines a boolean value.	Element
	Number		Defines a number value.	Element
	Literal		Defines a literal value.	Element
	Time		Defines a time value.	Element
	And		A composite for context elements.	Element
	Or		A composite for context elements.	Element
	Xor		A composite for context elements.	Element
	Not		A composite for context elements.	Element
	Include		Includes other context elements to the context definition.	Element
		contextElementId	Defines a context element that is included to the context definition.	Literal
Communication Actions	**Request**		Defines an action that requests a task.	Element
		taskId	Defines id for the task that is requested.	Literal
		[requestTimeOut]	Defines a time-out for the request.	Number
		[reprocess]	If has a value "true", a new task is started for the request.	Boolean
		[taskFactory]	Defines a task factory for the request.	Literal
	Modifier		Modifies the request before it is passed for the task factory.	Element
	Adaptor		Adaptor actions are notified about the responses of the requested task.	Element
		adaptorType	Defines the response type that the adaptor action will handle.	all | ready | update | failed
	Respond		Delivers the response of the task for the requestors.	Element
		responseType	Defines response type.	task_ready_response | response_updated | error_response | task_failed_response
	Refresh		Defines adaptation actions to do postprocessing related to the task.	Element
		[refreshController]	Define a plugin that sets the refresh attributes at runtime.	PluginDescription
		refreshTime	Defines when the defined refresh actions are executed.	Number
		refreshDelay	Defines how often the defined refresh actions are executed.	Number
		repeatCount	Defines how many times the defined refresh actions are executed.	Number
Control Actions	**Sequence**		Defines a sequential execution for child actions.	Element
	Exclusive		Defines an exclusive execution for child actions.	Element
		selectStyle	Defines how the child action is selected to be executed.	first | best
	Enumerate		Repeats the defined actions for all the child objects of a data object.	Element
		collection	Defines a name for the data that is enumerated.	Literal
	If		Define a condition, multiple if and else-actions, and a plugin that decides are the if or else-actions executed.	Element
		condition	Defines a condition for the if-else structure.	Literal
	Try		Define adaptation actions for exceptions. The exception is passed for the first catch action composite that is suitable for PC.	Element
Interactive Actions	**Accept**		Defines a structure that enables the user to control the adaptation. After the user has selected, either the accepted or rejected actions are executed.	Element
		acceptor	Defines an action to display UIs that enable the user to control the adaptation.	PluginDescription
		[acceptTimeout]	If defined, the accept actions are executed if the user does not reject the actions during the defined time-out.	Number
	Select		Defines a structure that enables the end-users to select between alternative adaptation actions. After the user has selected, the actions of the selected option are executed.	Element
		selectorAction	Defines an action that can present UIs enabling the user to select between alternative adaptation actions.	PluginDescription
		[selectTimeout]	If defined, the default actions are executed if the user does not select any option during the defined time-out.	Number

[AttributeName] = optional attribute

Figure 22. The refined meta-model introduces new action, content, and context elements for task-based composition.

4.7 Task-based speculative adaptation

4.7.1 Problem

Mobile usage is spontaneous and requires efficient adaptation methods that are capable of preparing applications parts for possible forthcoming processing contexts. Methods are needed to support *predictive and speculative composition of adaptive applications*. At the same time, it is important that the speculative adaptation is executed without disturbing the user of the application. For example, the response times of an application can increase if a lot of tasks are doing speculative adaptation in the background while the user is actively using it. This can be avoided by using the idle time of the application for speculative adaptations. The following states can be defined for an adaptive application utilising idle time for speculative adaptations (Figure 23):

- ***Not active state***. The application is not yet started.
- ***Adaptation state***. The application is adapted for the current PC.
- ***Active state***. The application is displayed for the user and is ready for use.
- ***Speculative adaptation state***. Speculative adaptation tasks are executed in this state. Prediction models can be updated, possible forthcoming PCs can be predicted, and application instances can be prepared for the predicted PCs.
- ***Idle state***. The application is not actively used in this state and speculative adaptations are not made.
- ***Stopped state***. The application is stopped in this state.

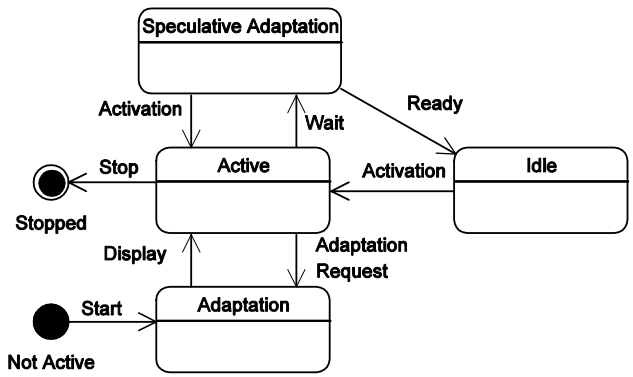


Figure 23. The states of an adaptive application.

In addition, there are the following kinds of transitions between the states:

- ***Start***. The adaptive application is started and adapted before use.
- ***Display***. The adapted application is displayed for the user.
- ***Activation***. The activation is made when the user starts to use the application or when the PC changes. The running speculative adaptations are suspended while the user is using the application.
- ***Wait***. Speculative adaptations are made while the application is not used.
- ***Ready***. ***Application*** is an idle state after speculative adaptations are ready.
- ***Adaptation Request***. If needed, adaptations are requested when the PC changes.
- ***Stop***. The application is stopped after a use.

The speculative adaptation requires:

- Actions that utilise *prediction models* and control speculative adaptation. Actions are needed to activate new speculative adaptations and to stop the previously started adaptations that are no longer needed.
- Methods that support of utilisation of application instances that are prepared for possible forthcoming processing contexts.
- Actions that update the prediction models and thus enable them to learn about user behaviour.

4.7.2 Solution

The task-based composition technique supports caching of context-sensitive task and application instances (see Section 4.5). Concurrent tasks can perform speculative adaptations in the background. For example, these tasks can fetch composition schemas, initialize task factories for these, and finally request new tasks to compose application instances for any possible forthcoming processing contexts. The initialized task factories and composed application instances are stored to the cache. As a result, the cached application instances can be utilised when the application is composed for a new PC. In addition, the cached task factories can be later used when application instances are composed for forthcoming PCs.

The prediction models can learn about the made adaptation requests. For example, these models can learn about user behaviour by recording request sequences during the usage of the application and predict adaptation requests that will be possibly made in the future. In order to make a speculative adaptation request better correspond to potential future contexts, the predicted request can define part of the PC values. In this case, new PC is constructed before the speculative adaptation task is requested. The new processing context is a clone of the PC, which is then modified with the context values defined in the predicted request.

The speculative adaptation actions can utilise the prediction models stored to the application environment, to update them, and thus enable them to learn about user behaviour. In addition, the composition schemas can define context-sensitive settings and speculative adaptation actions that will configure these prediction models to work better in various PCs.

If the context is changed, a lot of concurrent tasks may prepare application parts that are not suitable for current or possible forthcoming PCs. In order to minimise processing and to make adaptation for new contexts faster the unfinished tasks that are not suitable for the current or possible forthcoming PCs must be stopped and removed from the task cache. An unfinished task is stopped by preventing it from delivering new responses for the requestors. However, it must be noted that it is not always safe to stop task execution. For example, data might be corrupted if the task manipulating it is stopped at the wrong moment.

Actions can define specific points where it is safe to stop a task that is not suitable for the current or possible forthcoming PCs. As a result, it is possible to stop task execution before all the actions of the task are executed.

4.7.3 Summary of the solution

The task-based composition technique supports speculative adaptation and caching of context-sensitive tasks and application instances. Concurrent tasks can perform speculative adaptation and prepare application instances for the possible forthcoming PCs. Adaptation actions can define specific points where it is safe to stop a task that is not suitable for the current or possible forthcoming PCs.

4.7.4 Example

It is easy to extend or replace tasks with new ones. Figure 24 shows how a composition schema can be extended with a speculative adaptation task. The original "*Prepare Application"* task is renamed *"Original Prepare Application"* in the schema. The new "Prepare Application" task will request the original prepare application task first. The speculative adaptation actions of the task can be configured for different contexts. As a result, only speculative adaptation actions that are suitable for the processing context are executed. Learning actions can mutate prediction models. In order to keep the prediction models valid, the *prepare application* task is not cached but is executed always when requested.

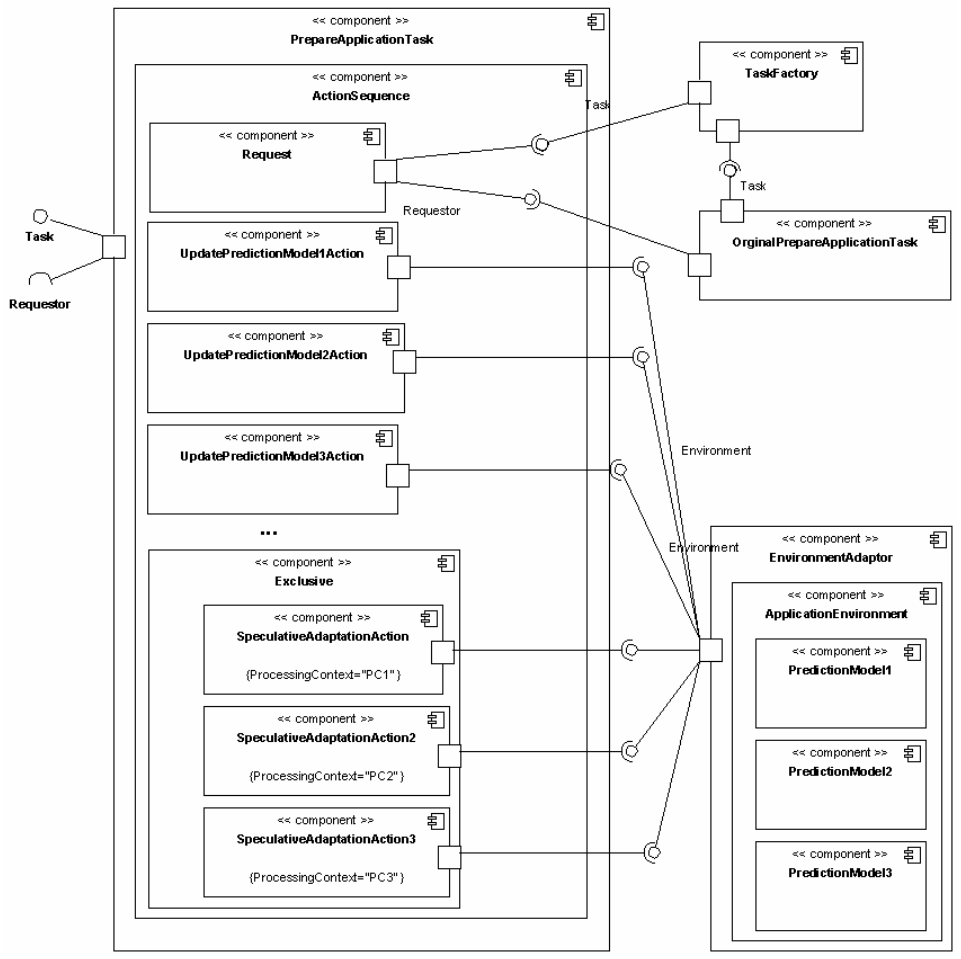


Figure 24. An extended composition schema for a task performing context-sensitive speculative adaptation.

4.8 The utilisation of the task-based composition technique

The task-based composition constitutes a foundation that provides a common mechanism to create and use tasks in dynamic component-based composition. It enables developers to define contexts-sensitive adaptation tasks that can prepare application instances for various contexts. Each task can have multiple actions and it can request other tasks, too. Thus the technique enables an application to

be adapted in many phases. To be practical and truly useful, tool support for the technique is needed. Such a tool could utilise the task-based composition methods and offer a useful set of tools helping developers to define composition schemas for various kinds of adaptive applications. A tool that facilitates implementation of XML-based composition schemas is discussed in Section 6.5.

The task-based composition technique enables developers to construct adaptive applications in the following phases:

1. Developers can implement composition schemas that define tasks and context-sensitive actions and their settings.
2. Executable adaptation code is attached to the composition schema by implementing various kinds of action plugins for tasks.
3. Developers can generate or implement test sequences requesting tasks for various contexts and run the adaptive applications composed with tasks in the simulator or in the actual runtime environment.

The task-based composition technique has four kinds of users shown in the UML use case diagram in Figure 25.

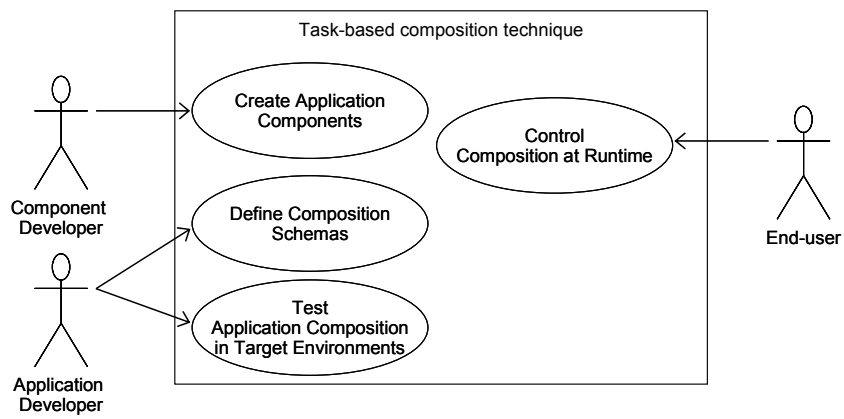


Figure 25. Use and the users of the task-based composition technique.

A component developer creates components for an adaptive application. An application developer defines task-based composition schemas and tests how the

composed adaptive application works in the target runtime environments. Finally, the user can control the composition at runtime.

To illustrate the use and advantages of the task-based composition technique, consider composition that is inside the application logic. This leads to a situation where the dynamic composition process has to be decided during the time when application components are developed. By composing an adaptive application with tasks, it is possible to clearly separate composition concerns from the application. It enables a separate developer to compose applications of components that other developers have implemented. The dynamic composition process can be improved afterwards and may be modified even at runtime. In addition, it may be possible to reuse composition schemas in different applications.

The task-based composition technique can be used in different application platforms. A context-aware application can have a part for sensing the context and adaptation requestors. For example, as proposed in Context Toolkit [DSA01], *aggregators* can collect context information coming from separate sources. A *requestor* can work as an aggregator that utilizes various kinds of context-sensing services and platforms, sense the changed processing context, and finally request a task to perform the adaptation actions. The actions of the task can prepare and compose application instances to the application and cache environments and so adapt the application for new PCs.

However, the context acquiring, processing, and predicting methods are not directly in the scope of this thesis and are therefore not discussed in detail. In this thesis it is assumed that the available requestors may request adaptation tasks for both the current and predicted processing contexts.

The task-based composition technique offers an empty skeleton for adaptive applications. It has only an application activator that requests an initialise task that may have various kinds of actions initialising the application environment when the adaptive application is started (Figure 26).

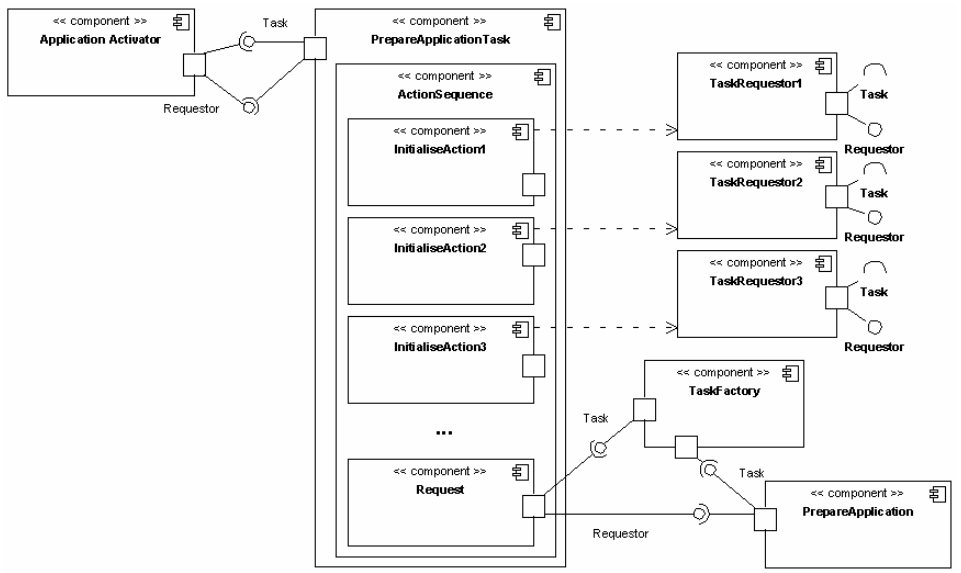


Figure 26. A composition schema for a task that constructs task requestors that may utilize separate context-sensing platforms.

The developers can modify composition schemas in order to make an adaptive application to better utilise the available (e.g. context-sensing) services and platforms. For example, new actions can be added to the initialise task that composes requestors to control the adaptation. For example, a requestor can invoke a task to perform adaptation actions if a context-sensing platform notifies it about the changed context.

4.9 Usage scenarios for the task-based composition technique

This section presents three usage scenarios for the task-based composition technique. Tasks can compose context-sensitive UIs for various contents and physical environments (Subsections 4.9.1 and 4.9.2) and perform speculative adaptation in order to make the usage of an adaptive application more fluent (Subsection 4.9.3). A short description, problem analysis, and task-based solution are provided for each usage scenario.

4.9.1 Task-based dynamic composition of content and context-sensitive user interfaces

Usage scenario

The small display of a mobile device set limitations for user interfaces and content presentations. For example, it may be difficult to present tables (e.g. timetables) and frames of the Web pages in the small display. Context-sensitive layouts and style sheets can improve the usability of Web pages. For example, content elements can be emphasized, hidden, and arranged in a specific way when the user is in a certain context.

Problem analysis

Wireless network connections are not typically as reliable as the fixed line connections. In addition, the data transmission rates are typically slower and the costs are often higher in the wireless networks. Thus it is important to minimize the usage of wireless networks.

The browsed documents can be cached to the local storage media. As a result, it may be possible to utilise these local copies of documents later, which can improve performance, reduce latency, and save network bandwidth and be cheaper for the users. However, it must be ensured that the cache does not have an outdated version of the document. HTTP/1.1 offers the means for ensuring that the cached documents are up-to-date [Luo98]. For example, it offers headers for performing *conditional* requests. The GET method uses the *Last-Modified header* that was received when the document was retrieved and stored to the cache. This value is sent in the *If-Modified-Since* request header. If the document is not changed, the server will return a not modified response status code and will not send the document content.

However, it is difficult to utilise a cached document if it is adapted for many (e.g. time, location, and user agent profile) context attributes. If the context is changed, the cached document is not necessarily suitable for the new context. In this case the needed document elements that are suitable for the new context must be fetched from the Web. Unfortunately, the data transmission is often chargeable in wireless networks. Furthermore, the network requests take time

and will delay the usage of the context-sensitive application. Thus it would be useful if a part of the adaptation could be made on the client-side. For example, documents may contain content elements for various contexts. As a result, the client-side can compose a presentation for the context by using the context-sensitive elements of the cached documents.

For example, in many cities (e.g. in Tampere) timetables for local busses are provided as HTML pages. However, it may be difficult to read these timetables from the small display of the mobile device. A content and context-sensitive user interface that presents only the departure times for the next busses that the user takes often can improve the usability significantly.

An XHTML document can contain meta-information that defines for which contexts the document and its particular elements are suitable. The *meta* element can define properties for the XHTML document whereas the *span* element offers a generic mechanism for adding structure to the document [DENW06]. The span element does not impose presentational idioms on the content but it can associate attributes with the content. The *content* attribute can associate metadata with the content of the span element. For example, span elements can define valid weekdays and times for the elements of a bus timetable (Figure 27).

```
<html xmlns="http://www.w3.org/2002/06/xhtml2/"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:dcterms=”http://purl.org/dc/terms/">
  <head>
      …
  </head>
  <body>
      <p>
            …
            <span content=”weekday=Monday-Friday;direction=tocity”>
                  <span content="title">
                        Bus 23 to City
                  </span>

                  <span content="time=10:00-10:59">
                        <str>10</str> 20 40 55
                  </span>
                  <span content="time=11:00-11:59">
                        <str>11</str> 20 30 52
                  </span>
                  …
            </span>
            …
      </p>
  </body>
</html>
```

Figure 27. Meta and span elements can attach meta-information to an XHTML document.

An XHTML document can be browsed with standard browsers that may ignore this meta-information or with specialised user agents that can utilise meta-information attached to content elements. For example, the user agent can display either all the contents of the XHTML document or then it can display only the information needed in the context.

The problems related to the described usage scenario are special cases of problems that the task-based composition technique helps to solve. Firstly, it must be possible to compose UIs in multiple phases for various content sources and contexts (relates to Sections 4.2 and 4.4) and to change the adaptation policies at runtime (relates to Section 4.6). Secondly, in order to improve the performance, methods are needed to improve the caching of content and context-sensitive instances (relates to Sections 4.3 and 4.5). Thirdly, it must be possible to fetch documents that are possibly needed in the future (relates to Section 4.7).

Task-based solution

Content and context-sensitive user interfaces can be composed with tasks. The server-side can provide a composition schema that defines adaptation actions that first download referred documents, later find correct content elements for PC, and finally compose an overall UI for the context and for the selected content elements. Content selection and UI composition actions can be configured with context-sensitive parameters that can define which kinds of content elements should be selected in various contexts and also configure the visual appearance of the composed UI elements.

The task-based composition technique supports both caching of tasks and application instances. As a result, applications can be composed in multiple phases where the already prepared application instances and responses of the previously executed adaptation tasks can be utilised in dynamic composition. Tasks can first download the document sources and then store parsed document sources to the cache. Tasks can later compose various kinds of context-sensitive user interfaces for the cached contents on the client-side without using the wireless network and without parsing document sources again. This all can make browsing more fluent in the mobile environment.

Tasks can compose user interfaces and content presentations of *glyphs*. A glyph is a flyweight component that can e.g. be a graphic or textual primitive or

a composite of objects [CaL90]. The Glyph base class defines a protocol for drawing; subclasses define specific appearances such as graphic primitives (lines and circles), textual primitives (characters and spaces), and composite objects (tilings and overlays). As a result, hierarchies of glyphs can define an appearance for an application.

The glyph protocol consists of *request*, *allocate*, and *draw* operations [CaL90]. Request defines the preferred screen space allocation of the glyph. Composite glyphs calculate their own requests from the requests of their components. Allocate tell the glyph how much space it actually got. Composite glyphs apportion their allocation among their components according to the requests of the components. Draw specifies the appearance of the glyph. Composite glyphs draw their components recursively.

A *frame glyph* can offer access to the root glyph and to the named glyphs of the presentation. As a result, separate glyph-based user interfaces can be composed to these named glyph composites. Figure 28 shows a frame glyph that defines glyph composites for time and bus timetable glyphs. Furthermore, it defines a default glyph composite for glyphs that are composed for other content types.

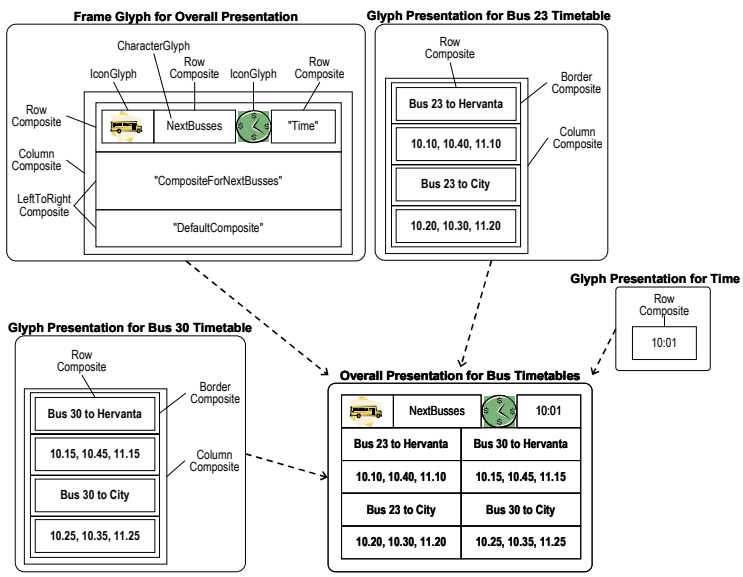


Figure 28. A content and context-sensitive UI for bus timetables.

A content and context-sensitive user interface is composed of glyphs in the following phases.

1. **Selection of content sources and composition schemas**. The user may have a list of Content Source Descriptions (CSDs) that can describe contents available on the Web. CSD can define a source, type, and composition schema for a content document. The source defines a Web address for the document source whereas composition schema defines a task that is able to compose context-sensitive UIs for the content elements of the document. The user can select interesting contents to be displayed in overall UI. Furthermore, he or she can select specialised composition schemas for content sources and for overall UI.

2. **Fetching of contents and composition schemas to the cache.** Contents are fetched and stored to the cache. At the same time, new task factories are initialised with the fetched composition schemas and stored to the cache. The cached task factories are later requested to execute tasks that will compose context-sensitive UIs for the selected contents.

   The context comparators control the caching of the contents. For example, they can use the meta-elements of a parsed XHTML document, which can define when the document has expired or for which contexts the document is suitable. If the document has expired, it is removed and a new one is fetched from the Web and stored to the cache.

3. **Composition of a frame glyph for overall UI.** A task composes a frame glyph for overall UI. The frame glyph can have named glyph composites for glyphs that are composed for certain content types. Furthermore, it may define a default composite for glyphs that are composed for content types that are not directly supported by the frame glyph. The task delivers a frame glyph for the overall presentation in its response. As a result, actions of another task can compose the glyphs of other content types to the overall UI.

4. **Composition of UIs for CSDs.** The cache has a task factory for each composition schema. A task factory initialised with a correct composition schema is called to execute a task that will compose a glyph-based context-sensitive UI for the content source defined in CSD. The task will deliver the UI in its response, which is later composed to the frame glyph of the overall UI.

For example, a bus timetable can be an XHTML document, which may provide various kinds of meta-information and context-sensitive content elements. Meta-information can be attached to the elements of the bus timetable with span elements. The content attribute of a span element defines direction (to the city or away from the city), time, and weekday for the content elements of the timetable. As a result, the actions of a UI composition task can use this meta-information and search the most suitable content elements for PC. Actions can later compose glyph-presentations for the selected elements.

5. **Refresh for overall UI.** The tasks can be requested to prepare new UI if the context is changed. Furthermore, refresh actions can utilise the metadata of the document source and update composed UI periodically. For example, if the fetched contents are expired, the refresh actions can fetch a new content document to the cache, compose UI for it, and replace old glyph-based UI of the expired content with the new one in overall UI.

   The refresh action may utilise the metadata defined in the (e.g. XHTML) document source. It can fetch a new document if the cached document has expired and compose a new presentation for it. The time is shown in the bus timetable example UI. The refresh action can update the UI once in a minute and so ensure that the correct time is displayed for the user.

A task that will compose overall UIs for various content elements is presented in Figure 29.

The task that will compose a frame glyph for the overall presentation is requested first. Then, a task is requested to compose glyph-based UIs for CSDs. Finally, the adapt actions of the request compose UIs of CSDs to the named glyph composites or to the default composite, if a named glyph composite is not found for the UI of CSD. The adapt actions can replace the old glyph with a new one if the UI of CSD is updated. The composed UI will be refreshed periodically. For example, the UI can be recomposed if the contents composed to the UI have expired.

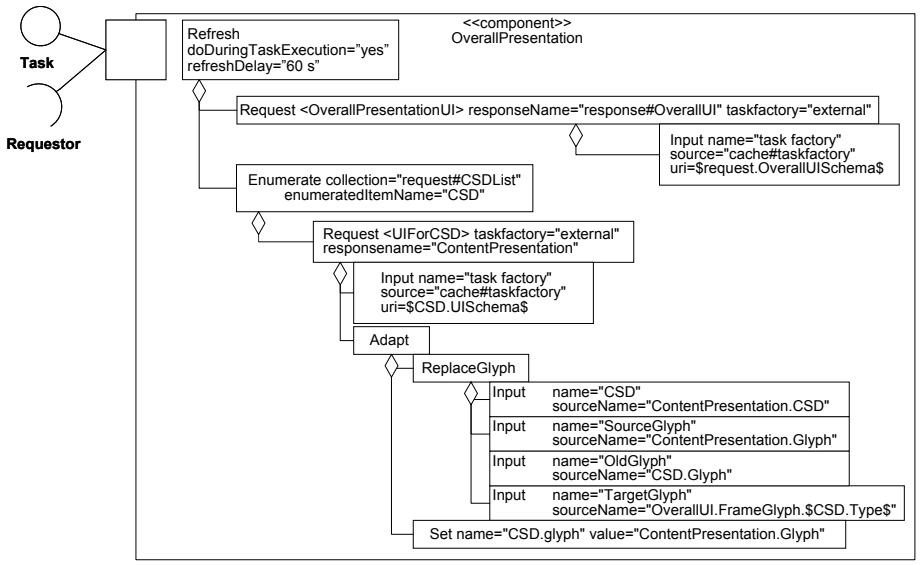


Figure 29. A task that composes an overall UI for various contents.

4.9.2 Task-based dynamic composition of context-sensitive user interfaces of physical environments

Usage scenario

In the future there will be more and more physical devices and objects that are connected to the Web. Many kinds of devices and objects (e.g. TV, radio, video, desktop computers, or other appliances) may be controlled through (e.g. Bluetooth) Access Points (APs) and with UIs that are made for these different physical objects (Figure 30).

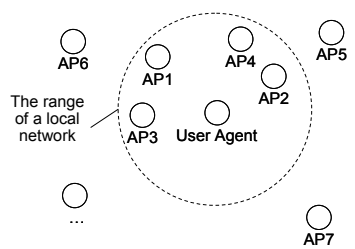


Figure 30. A physical environment can offer access points for various kinds of objects that can be observed and controlled with Web UIs.

As a result, a physical environment can have a lot of objects that can be controlled through access points and with separate UIs that are made for different physical objects. The user may be interested in observing and controlling only specific physical objects that are inside the range of a local network. It should be possible for the user to select certain objects from a physical environment and to use these nearby objects with a single UI.

Problem analysis

Mobile devices have typically only limited input and output capabilities. It is important to adapt UI to contain only information and controls that are needed in the physical environment and context. For example, if the light switches are hidden during the day time in overall UI, it is easier for the user to control the other home appliances with the UI running on a mobile device. This all sets requirements for a technique that can compose UIs of physical objects into a single UI that is adapted for the context.

The task-based composition technique helps developers to solve the problems related to the described usage scenario [Pal07]. Firstly, it must be possible to compose context-sensitive UIs for a physical environment and its objects in multiple phases (relates to Sections 4.2 and 4.4) and to change the adaptation policies at runtime (relates to Section 4.6). Secondly, in order to improve the performance, methods are needed to improve the caching of application instances that are needed in UIs (relates to Sections 4.3 and 4.5). Thirdly, it must be possible to compose UIs that are possibly needed in the future (relates to Section 4.7). Tasks support dynamic composition of UIs for combinations of services and improve performance when UIs are adapted for new contexts and physical environments.

Task-based solution

Figure 31 shows a frame glyph that defines glyph composites for glyphs related to time, location, lights, and TVs and a default glyph composite for glyphs that are composed for other contents or physical object types.

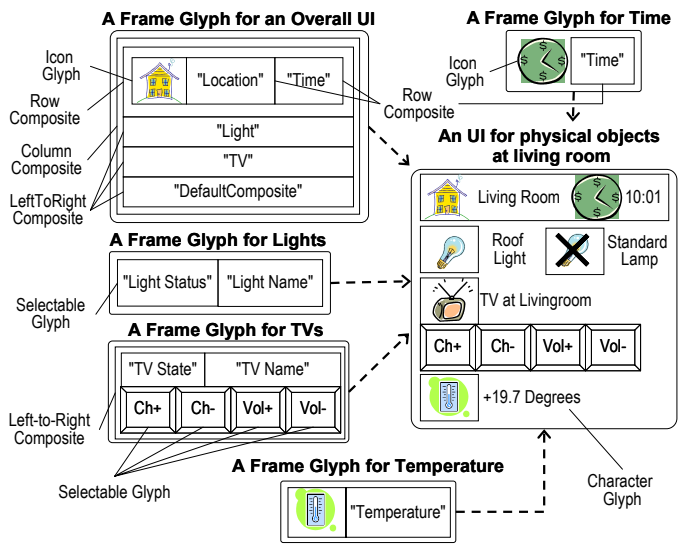


Figure 31. UIs of a physical environment can be composed of glyphs.

Tasks can compose a glyph-based UI for a physical environment in the following phases (Figure 32).

1. **The Physical UI (PUI) task requests a task to compose a frame glyph for the overall UI.** A *frame glyph* defines a layout for the overall UI. It offers access to the root glyph and to the named glyphs of the presentation. As a result, separate glyph-based UIs can be composed to these named glyph composites.
2. **The PUI task requests a task to fetch the Physical Object Descriptions (PODs) from access points (APs).** A POD defines a name, type, attributes for a physical object and an address for a service (*ObjectURI*) that enables the user to control and observe the object. It can also define a source for a composition schema (*UISchema*) that describes a task that is capable of composing a UI for the POD.
3. **Composition of UIs for the Physical Object Descriptions (PODs).** The PUI task will fetch the UISchemas for found PODs, initialise task factories for them, and finally request their tasks to compose UIs for PODs that are finally composed to the named glyphs of the overall UI.

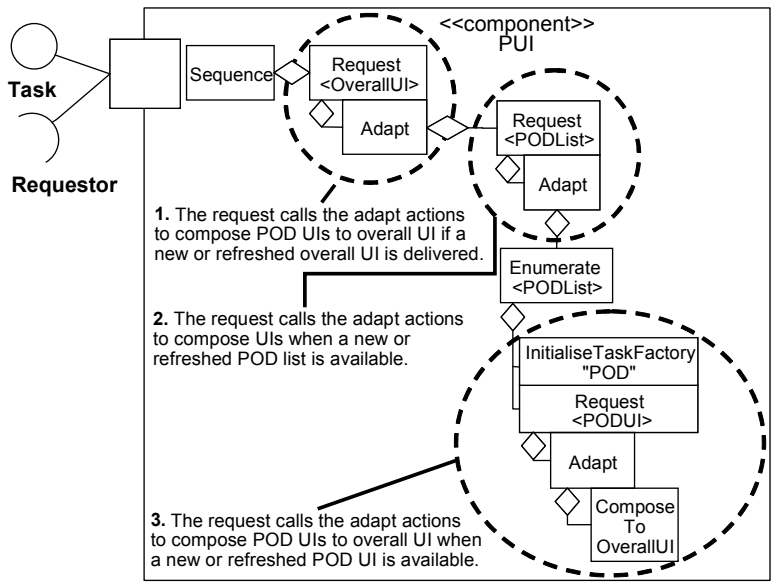


Figure 32. A task that composes an overall UI for physical objects in multiple phases and performs reactive and proactive adaptations for it.

4. **Reactive and proactive adaptations for the UI.** The context and physical environment can change rapidly. The *refresh* action, *request-adapt* structure, and caching of tasks and content and context-sensitive instances provide an efficient way to make adaptations in the UI. Firstly, reactive adaptation tasks compose UIs for new requests and contexts. This is made efficiently because the cached application instances are utilized in dynamic composition. Secondly, tasks can do multilevel proactive adaptations. The refresh actions can update the response of a task when the physical environment or context is changed or update the response periodically. For example, the refresh actions can update the overall UI, POD list, or only a UI of a single physical object.

   The following steps are made if the overall UI is updated (Figure 32). (1) The notified adapt actions request a task to deliver a POD list first. It will fetch new PODs from the available APs only if a valid POD list is not available in the cache. (2) The adapt actions will request tasks to compose UIs for PODs. Tasks will compose new UIs only if suitable UIs are not available in the cache. (3) Finally, the adapt actions will compose the UIs to

the overall UI. Only steps 2 and 3 are made, if a POD list is updated and only step 3 is made, if a UI of a single physical object is updated.

However, usability problems may arise and the inputted information can be lost if a UI is replaced with a new one while the user is actively using it. The requestors and refresh actions can observe the activity of the user and use the idle time for adaptations. As a result, the UI can be adapted without disturbing the user.

5. **Speculative adaptation** tasks can prepare new task factories and compose UIs for the next locations to which the user will possibly go next. The task factories and UIs are cached. As a result, cached UIs are fast displayed for the user when he or she arrives in a specific physical environment. In addition, the cached task factories can be later utilized when UIs are composed for new PCs.

Composition schemas can describe tasks to compose default UIs for PODs to enable the user to use the standard features of certain kinds of physical objects. Task factories can be initialized for these default composition schemas and cached. As a result, default UIs can be fast composed and displayed for the user.

Tasks that are capable of composing UIs for specific physical objects or environments can be provided, too. Specialized UIs can improve the usability and enable the user to utilize features available in specific appliance (e.g. in video recorder) models only. The background tasks can download composition schemas for specialized UIs, initialize task factories for them, and request tasks to compose specialized UIs for PODs when the user is using the default UIs. However, usability problems may arise if UI is replaced with a new one when the user is actively using it. The user-directed adaptation can solve this problem. A default UI can offer a control (e.g. a button) that enables the user to request a task that will replace the default UI with a specialized one.

The task-based composition technique facilitates the dynamic composition of UIs of physical environments and supports client and server-side adaptation. A composition schema defines tasks and context-sensitive adaptation actions, parameters, and contents for them. As a result, it is possible to compose context-sensitive UIs for both physical objects and environments.

A physical environment and its objects may change constantly. This requires dynamically changing UIs that are adapted for the physical environment in question. Unlike many mark-up languages, a composition schema does not necessarily directly define UI or content elements but describes how a UI should be composed for physical environments, objects, or contents in various contexts. For example, if a room has five light switches, UIs of different switches can be composed on the client-side by using a single composition schema and without downloading separate UIs for various light switches. The composition schemas can be downloaded either from the Web or from the local cache. For example, if the context is changed, the UI can be adapted for the new context on the client-side. In the best case, if the needed resources are in the cache, the UI can be composed without causing network traffic at all. Only, the list of PODs of the objects of the physical environment needs to be acquired.

If needed, the XML-based composition schemas can be modified or replaced with new ones. For example, the user may fetch a new composition schema from the Web that can compose better UIs for certain kinds of physical environments. In addition, it can introduce a new layout and overall UI for physical environments or replace default UIs of the physical objects with new ones. For example, it can use a composition schema that will make specialised UI for a certain video recorder model. As a result, the user may utilise the features that are available in that specific video recorder model only.

4.9.3 Task-based speculative adaptation

Usage scenario

If an application is composed of local components, the composition time can be relatively short and may not have such a great effect on the usability. The composition time is typically longer (e.g. more than ten seconds) if non-local resources and components are used in adaptation. In order to enable seamless adaptation, the tasks can be requested to prepare application instances for potential future contexts in the background. For example, tasks can prepare application instances related to locations at home and save them to the cache in the background. As a result, the application can be fast composed of the prepared instances when the user comes to a certain location at home.

Problem analysis

A physical environment may offer various kinds of networks for mobile users. For example, in the office the user may use a Wireless Local Area Network (WLAN) connection, whereas while travelling there can be only a General Packet Radio Service (GPRS) connection available. In addition, the user may have a home automation system that can be used through Bluetooth access points. The characteristics of these wireless networks vary a lot (Figure 33). For example, pricing, bandwidth, and coverage are very different in various networks. Thus in order to make the usage of Web applications more fluent and cheaper for nomadic users, an application should adapt to use the network that is most appropriate for the context.

Network	Bandwidth	Coverage	Pricing model	Prices 12/2004	Possible use in business applications
GSM-data	9.6-14.4 kbps	GSM-coverage	Time	0.11-0.16 € per minute	WAP browsing, Email, M2M
HSCSD	Max 57.6 kbps	GSM-coverage	Time	0-22-0.32 € per minute	Email
GPRS	30-40 kbps	Almost GSM-coverage	Traffic	1.5-1.8 € per megabyte	Content services, email, MMS, WAP browsing, M2M
EDGE	Max 110 kbps	Major cities	Traffic	1.5-1.8 € per megabyte	Content services, email, MMS, WAP&WEB browsing
UMTS	Max 384 kbps	Major cities	Traffic	1.5-1.8 € per megabyte	Content services, email, MMS, WAP&WEB browsing, remote work
WLAN	512kbps – 4Mbps	Hotspots	Fixed/traffic/time	0.25 € per minute up to 90 € per month	Internet browsing, email, remote work
Bluetooth	Max 700 kbps	Hotspots	Service	-	Local (range up to 100m) services

Figure 33. A summary of the connectivity technologies [AAH05].

The following example shows how the task-based composition technique can be utilised in the applications using network connections. Consider a situation where the user downloads a Word document to the mobile device at work. Then, the user may need a map service and driving instructions on a work trip that helps him or her to find specific locations in the countryside. Finally, after the

user comes home, he or she can download the user interfaces of the home automation system and use the available services with the mobile device. The sizes of the downloaded files used in the usage scenario are given in Figure 34.

Web Resource	Size [kByte]
Word Document	1000
Map Service	500
Home Automation UIs	300

Figure 34. The sizes of the files used in the case study.

For example, the user may use networks A, B, and C (Figure 35).

Network	Speed [kByte/s]	Cost [Euros/Mbyte]	Coverage
A	50	0,1	Office
B	4	1,5	Everywhere
C	30	0	Home

Figure 35. The characteristics of networks A, B, and C.

Network A can be available at the office, network C at home, and B everywhere. The simplest solution is to make the application to always use network B, when the total download time is 450 seconds and costs are 2.7 euros (Figures 36 and 37). The download time is 36 seconds and costs are 18 cents if all the resources could be downloaded over the network A. Network C does not cause any costs for the user but it takes more time (233.3 seconds) to download the resources over the network C. Unfortunately, networks A and C can only be used in certain locations.

In order to make it more fluent and cheaper to use services available on the Web, the combination of these three network technologies can be used. The Word document can be downloaded by using the network A at the office, maps can be downloaded by using the network B during travel, and finally the user interfaces of the home automation system can be downloaded by using the network C. As a result, the download time is 155 seconds and costs are 85 cents, if these resources are downloaded over networks A, B, and C.

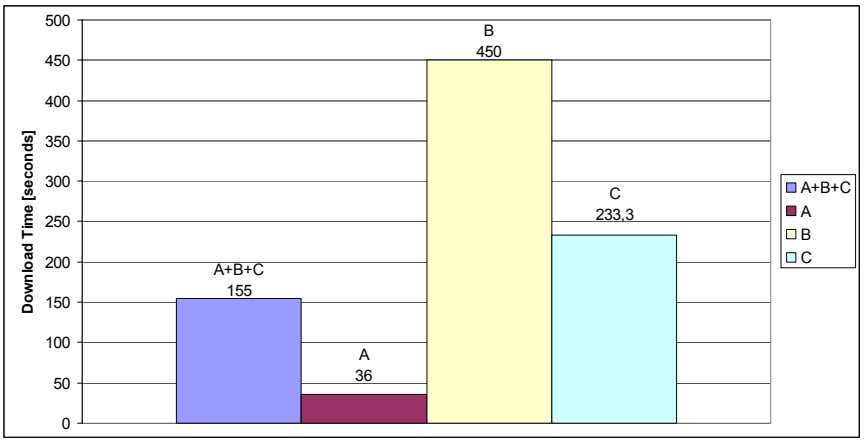


Figure 36. Download time for the resources transmitted over networks A, B, and C.

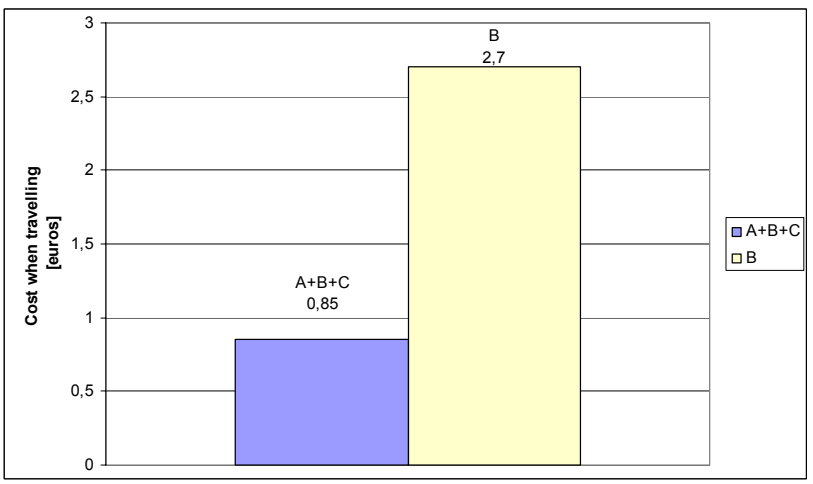


Figure 37. Costs for the resources transmitted over networks A, B, and C and over network B.

Speculative adaptation can be used to improve performance and reduce costs. For example, the application can use the calendar information and so it can know that the user will have a business trip. As a result, the application can conclude that the user may possibly need a map for finding the locations en route. In addition, the user may need the user interfaces of the home automation system after she or he arrives at home in the evening. Thus the application can

download these files by using network A when the user is at the office and is not actively using the mobile device. As a result, the download time is 36 seconds and costs are 18 cents if the resources are downloaded over network A.

In the previous usage scenario it is assumed that a prediction method exists that is able to predict the required speculative adaptations perfectly. Unfortunately, the prediction accuracy is typically less than 100% in the real world (see Section 2.2.4). This will cause vain speculative adaptation. The speculative adaptation is a reasonable solution only if it benefits the user of an adaptive application. Thus it is required that:

$$cost_{spec} < cost_{normal}$$

where

, $cost_{spec}$ is the cost of correct and vain speculative adaptation.

, $cost_{normal}$ is the cost without speculative adaptation.

In the usage scenario there are calculated download times and prices when various contents were downloaded over different networks. The following kinds of equations can be written for the usage scenario and for $cost_{spec}$ and $cost_{normal}$:

$$cost_{spec} = S*C_S + (1-\mu)*N*C_N$$

$$cost_{normal} = N*C_N$$

where

, μ is the prediction accuracy.

, S unit cost in the network where the speculative adaptation is made.

, C_S= Size of the content downloaded in the speculative adaptation.

, N unit cost in the network where the normal adaptation is made.

, C_N= Size of the content downloaded in the normal adaptation.

By using these equations, we can calculate the minimum requirement for the prediction accuracy μ ($0 \le \mu \le 1$):

$$cost_{spec} < cost_{normal}$$

$$\Rightarrow S*C_S + (1-\mu)*N*C_N < N*C_N$$

$$\Rightarrow \mu > \frac{S*C_S}{N*C_N}$$

The equation will look like this if we assume that the size of the content downloaded is the same in the normal and speculative adaptation ($C_S=C_N$).

$$\mu > \frac{S}{N}$$

We can use this equation and calculate the required prediction accuracy μ for the networks used in the usage scenario (Figure 38). For example, the speculative adaptation will decrease the download time if the prediction accuracy is more than 8% and make the content download cheaper if the prediction accuracy is more than 6.7% when the user moves from network A to B.

Network at Begin	Network at End	Required Prediction Accuracy (Speed)	Required Prediction Accuracy (Price)
Network A	Network B	8,0 %	6,7 %
Network A	Network C	60,0 %	-
Network B	Network A	-	-
Network B	Network C	-	-
Network C	Network A	-	0,0 %
Network C	Network B	13,3 %	0,0 %

Figure 38. Required prediction accuracy μ when the user moves between networks A, B, and C.

It must be noted that also other issues than prediction accuracy affect the costs of speculative adaptation. For example, the size of available memory can reduce content caching. In addition, the contents that are not needed must be later removed from the cache. This causes additional processing and can increase delays related to speculative adaptation. These issues were not considered in the previous calculations.

The task-based composition technique helps developers to solve problems related to speculative adaptation. Firstly, it must be possible to open connections and to fetch contents in various contexts (relates to Sections 4.2 and 4.4) and to change the adaptation policies at runtime (relates to Section 4.6). Secondly, in order to improve performance, methods are needed to improve the caching of context-sensitive documents (relates to Sections 4.3 and 4.5). Thirdly, it must be possible to prefetch contents that are possibly needed in the future (relates to Section 4.7).

Task-based solution

Tasks can perform speculative adaptation and thus improve performance and reduce costs. Different kinds of speculative adaptations may be needed in various contexts. For example, sometimes it may be possible to use prediction methods whereas sometimes it is only possible to request speculative adaptations randomly for possible forthcoming processing contexts. The task-based composition technique enables developers to attach various kinds of speculative adaptation mechanisms to the existing composition schemas of adaptive applications. The composition schemas can be extended with context-sensitive actions performing speculative adaptations. As a result, the most appropriate speculative adaptation actions can be executed in different contexts.

5. Implementation issues

Mobile usage is spontaneous and applications should be fast to install, start, and use in the devices with limited memory and processing power and in the wireless networks offering typically less bandwidth than wired connections. This all sets requirements for implementations that support task-based dynamic composition of adaptive applications. Firstly, it is important that using the task-based composition technique does not considerably increase the size of the installation package of an adaptive application. Secondly, it is important that the usage of the technique does not considerable increase the ready-for-use time of an adaptive application. Thirdly, in order to be usable in various kinds of mobile devices, it is important that using tasks does not cause a considerable processing overhead. These issues are discussed in Sections 5.1 and 5.2.

5.1 Execution of adaptation tasks

Concurrent composition is needed in many adaptive applications. For example, asynchronous content fetch is a commonly used technique to improve the efficiency of browsing applications. An HTML page can have several references to image, text, and video resources. The loading time for a single browsed resource is the total time spent on connection opening, request sending, and response receiving (Figure 39).

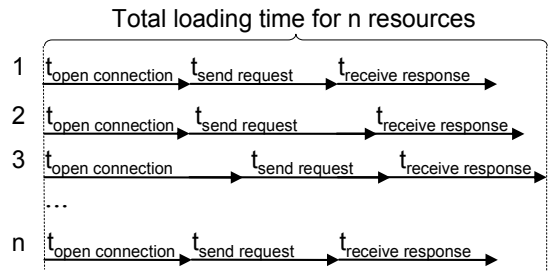


Figure 39. Total loading time for n resources loaded asynchronously over multiple connections.

In asynchronous content fetch separate threads can open multiple connections and fetch resources simultaneously, when the perceived load time is the

maximum of the load times of the resources [Luo98]. In addition, because content fetch is done in separate threads, it does not block the usage of the browser [CDK00].

The parts of an application can be replaced with new ones for adapting it for new contexts. For enabling fine-grained adaptations, the application can be composed of small parts. As a result, it may be easier to utilise the previously prepared parts when the application is adapted for new contexts. However, it must be noted that those actions modifying instances of the adaptive application will lock those instances and so may possibly block the use of the application. Thus it may be better if the actions of tasks do not modify the instances used currently in the application. Instead, actions can utilise instances when they are building new instances for possible forthcoming PCs. New instances can be saved to the cache and may be possibly composed to the application in the future.

Computer configurations typically only have a single CPU. The illusion of concurrency can be made by executing multiple threads on a single CPU. The *scheduling* defines the execution order for the threads. For enabling concurrent execution for tasks, a task is executed in its own thread if an asynchronous request is made. Unfortunately, constructing and starting new threads can cause overhead. It can be minimized by using a pool for the processing threads [Goe02].

Especially, in those situations where the context is changing very rapidly, a lot of tasks may be started to prepare application instances for various PCs. This may decrease the usability of the adaptive application. The application can be slower to use and the memory consumption can increase, if many concurrent tasks requiring a lot of processing are executed in the background.

In order to be usable in various kinds of mobile devices, it is important that using the task-based composition technique does not cause a considerable processing overhead. We made the following test preparation and measurements with the Java Mobile Information Device Profile (MIDP) enabled Nokia 6630 mobile phone in order to evaluate the amount of processing overhead in a case where multiple tasks participate in processing. The Java Virtual Machine (JVM) implementation, thread scheduling model, thread priorities, and processes running in the background may have a great effect on the processing time of

a single thread. Thus it must be noted that these results cannot be directly generalized for all runtime environments.

In order to measure the processing overhead related to tasks, we executed same calculation loops with threads and tasks. We implemented a composition schema (Figure 40) that forms a chain of tasks (like in Figure 11) in which each task (except the last one) modifies a request and later adapt the response of the requested task. The request modifier adds the value of a *request index* parameter by 1 first and attaches it to the new request. A new calculation task will be requested until the request index value has exceeded the defined value. As a result, only a defined number of tasks will be executed. The request action has an adapt action that takes the result of the received response as an input, performs a calculation loop, and finally saves the result of the calculation loop to the request.

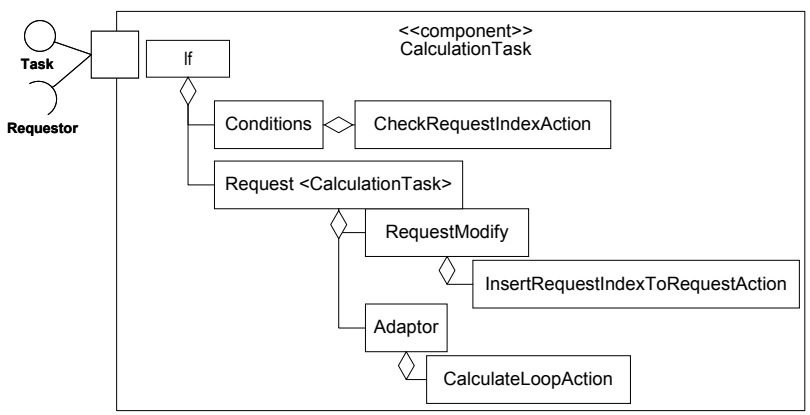


Figure 40. A composition schema for the chain of calculation tasks example.

The Java implementation of the task-based composition technique, called TaskCAD, (described in Chapter 6) was used in the measurements that were made in the Nokia 6630 device. One million calculations were made in a single calculation loop. In both cases a thread pool that had 100 Java threads was used. In the thread-based measurement, n number of threads processed the same calculation loop sequentially. The tasks were not cached in the calculation task

measurement. As can be seen in Figure 41, a linear dependency exists in both cases between the number of calculation loops and execution time.

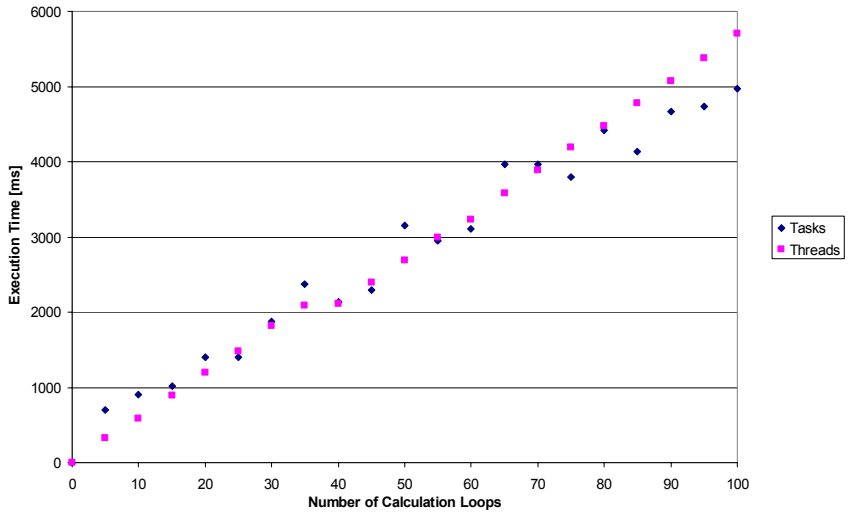


Figure 41. A processing time for calculation loops.

However, although more instances (e.g. task requests, responses, and action events) are composed in the task-based processing, the execution times are not considerably longer. However, it must be noted that the proportion of the processing overhead will increase if tasks are used for very small processing actions. For example, if a single calculation is made with a task, a calculation loop will make 1 million task requests. As a result, a huge number of requests and responses will be made that can increase the processing time drastically.

A mobile device may offer only a limited number of threads for an adaptive application. In this case the synchronization can be difficult if concurrent tasks are modifying shared objects. For example, the discussed chain of calculation tasks example does not work if threads are not available for all tasks. Tasks make synchronized task requests and then wait that a thread executes the requested task. If new threads are not available, the task will not be executed. As a result, execution will be blocked until the request time-out is expired. It should be noted that it is the concern of a requestor to define a long enough time-out

value to the task request. If the time-out is too short, the requestor (e.g. a task) will continue its execution before the requested task has delivered its response.

5.2 A generic structure for different data types

Mobile usage is spontaneous. Thus it is important that starting of an application does not take too much time. The ready-for-use time is longer for a Java MIDP application, if a lot of Java classes are loaded while the application is started. We make measurements in order to evaluate how much does the number of Java classes affect the ready-use-time in Nokia 6670 and Nokia 6630 mobile phones (Figure 42).

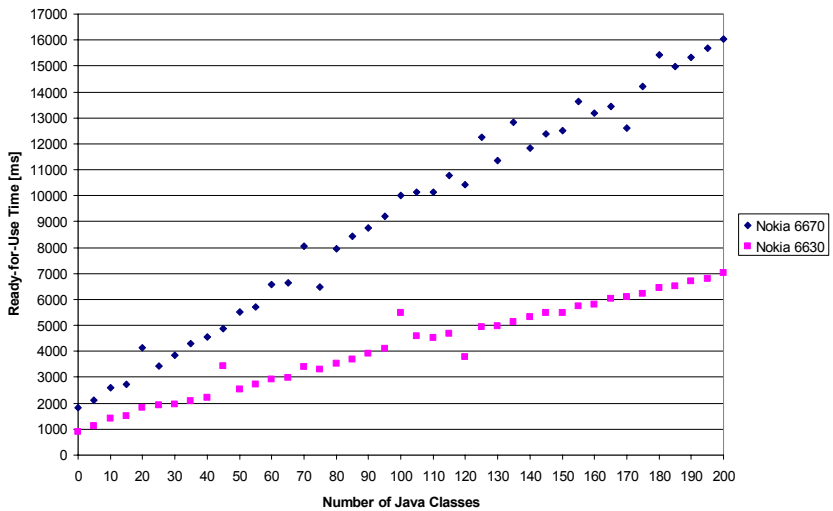


Figure 42. The ready-for-use time in Nokia 6670 and 6630 mobile phones for an application having various number of Java classes.

The installation package contained a total 200 generated Java classes. We implemented a Java MIDlet that shows a form first and then downloads a defined number of empty Java classes. Finally, the time that took to load those classes was shown in the form. As can be seen in Figure 42, the number of classes has a great effect on the ready-for-use time. The delay is over 10 seconds in the Nokia 6670 device when the number of loaded classes was 100 or more.

The task-based composition technique supports *lazy initialisation*. The classes defined in the composition schema are not loaded and application instances are not initialised before they are needed. This shortens the ready-for-use time of an application. However, it must be noted that the implementation of the task-based composition technique may increase the number of Java classes and interfaces, which may in turn extend the ready-for-use time. Thus in order to ensure that the ready-for-use time would not be too long, it is important that the implementation of the task-based composition technique does not contain too many Java interfaces and classes.

In order to minimise the number of required Java classes and interfaces, we followed the principles of the reflection architectural pattern [BMR+96] and implemented a generic structure for various kinds of data that is used in task-based composition (Figure 43).

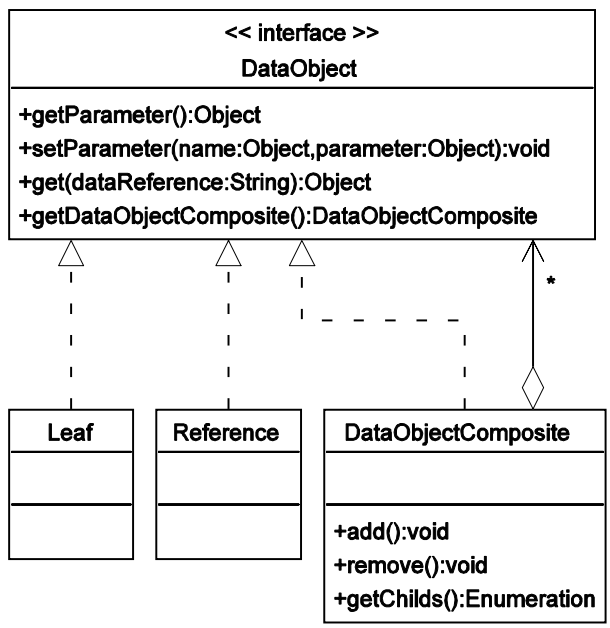


Figure 43. A generic structure for data used in task-based composition.

The data object structure is based on the Composite pattern [GHJV95] and offers implementations for three kinds of data objects. A data object can be a *leaf*,

a *composite* of data objects, or a *reference* to other data object. It can define parameters and a composite data object can contain child objects, too.

Adaptation actions can use data objects for information exchange. They can compose both parameters and child objects to data objects and utilize existing data objects in adaptation. It is possible to refer to parameters and child objects with a data reference formatted as follows:

DataRef = [#InstanceName][.ParameterName][DataRef].

For example, this enables the input and output mappings defined in a composition schema to refer to specific parameters and named child objects of the data object instances available at the runtime environment.

TaskCAD relies strongly on data objects. For example, the data related to composition schemas, requests, PCs, and responses and also events passed for adaptation actions are presented as data objects. This decreases the size of the installation package, because separate interface and class implementations are needed only for the functionality of the task-based composition technique. Currently, the Java implementation of the task-based composition technique contains 13 interfaces and 25 classes. For example, loading of these interfaces and classes took 5900 ms in the Nokia 6670 device and 2100 ms in the Nokia 6630 device.

The ready-for-use time can be shortened if a lot of functionality can be packed in a single class. In order to demonstrate this, we generated 100 action plugin classes that have only a single method. The load and initialisation of these classes took 10500 ms in the Nokia 6670 device and 4500 ms in the Nokia 6630 device. Then, we implemented an action that has 100 adaptation methods and a method that will select the correct adaptation method for a name and then will call it to perform the required adaptations. The load and initialisation of that class took only 350 ms in the Nokia 6630 device and 1100 ms in the Nokia 6670 device. The task-based composition technique makes it possible to combine several adaptation actions in a single action plugin. The composition schema can define the name for the adaptation action. As a result, it is possible to direct the request for the correct adaptation method when an action plugin is called to handle an action event.

Compared to a pure Java implementation, the usage of the task-based composition technique can increase the size of the application installation package. Firstly, interface and class implementations are required for the technique. The compiled size of the interfaces and classes of TaskCAD is currently 120 Kbytes. Secondly, an XML parser is needed because the composition schemas are defined with XML, which may also increase the size of the installation package. However, it must be noted that an XML parser is needed anyway in many applications in the mobile environment. Especially Web applications and standards based often on XML. A kXML 2 parser was used in XML parsing. Its compiled size is 31 Kbytes.

6. TaskCAD: An implementation for the task-based composition technique

The Java reference implementation of the task-based composition technique, called *TaskCAD*, offers core interfaces and classes that are capable of executing adaptation tasks with threads. Furthermore, it offers an interpretable Domain Specific Language (DSL) [DKV00] for composition schemas that enables developers to define tasks and context-sensitive actions, parameters, and content elements.

The TaskCAD provides an XML-based language for composition schemas which is discussed briefly in Section 6.1. The core implementations of TaskCAD are able to execute composition scripts and to use Java-based actions in composition. The core implementations of TaskCAD are discussed in Section 6.2. TaskCAD provides execution structures for adaptation actions and implementations to support caching of context-sensitive tasks and instances. These are discussed in Sections 6.3 and 6.4. Furthermore, an XML-editor is provided to facilitate construction of composition schemas which is discussed briefly in Section 6.5.

6.1 An XML-based language for composition schemas

The task-based composition language offers elements that enable developers to describe composition strategies for adaptive applications, to use various kinds of components in task-based composition, and to define new execution structures for adaptation actions. The XML-based language for composition schemas is based on the meta-model of the task-based composition language that was discussed in Section 4.6. The Document Type Definition (DTD) defines elements, attributes, and structure for the language. For example, XML editors can use the DTD and ensure that the required elements and attributes are defined in edited composition schemas.

The XML-based language of TaskCAD enables an application developer to create a composition schema that will generate a composition script that will finally configure the task factory (Figure 44).

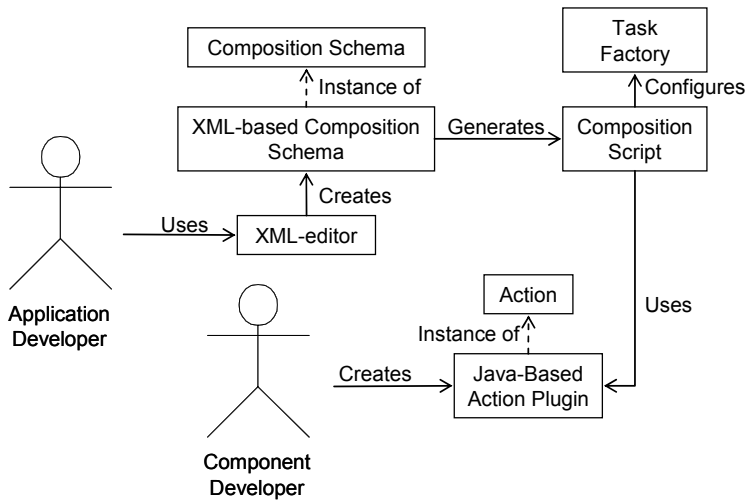


Figure 44. TaskCAD facilitates application and component developers to implement Java-based adaptive applications.

6.2 Task factory and composition schemas

TaskCAD offers ready-made Java implementations for a task, task factory, and composition schemas. The task factory composes tasks for requests and executes them with threads. In order to minimise the overhead related to constructing and starting threads, the task factory has a pool for execution threads. The size of the thread pool is set dynamically. A defined amount of threads is started at the beginning. If needed, more threads are started later to do the processing.

Composition schemas have an important role in task-based composition. They are used in task-based composition with the following steps:

1. **Composition of a data object presentation for the textual composition schema.** The reference Java implementation uses a kXML 2 parser in order to compose a data object tree for the composition schema described with XML. The tree can contain *leaf*, *composite*, and *reference* data objects.
2. **Attaching context comparators to the context-sensitive parts of a composition schema.** It is possible to attach functionality to the data objects. For example, context comparators are constructed and attached to

the context-sensitive elements of the composition schema and thus it is possible to filter correct data objects for certain PC. The source data object tree is traversed and suitable data objects are selected for PC by using the context comparators. If a context comparator is defined, it is called to calculate the Suitability For Context (SFC) value for the data object and PC. Finally, the data objects that are suitable for PC are selected and composed to a data object tree that contains correct data objects for certain PC. For example, this feature is used when the context-sensitive settings and content elements are fetched when an adaptation action is executed for certain PC.

The named context elements of a composition schema can specify acceptable PC values for the context-sensitive elements of the schema. TaskCAD presents context elements as a data object tree that defines a data object for each context element and can also contain data object references that will *include* other named context elements to the tree. TaskCAD offers a ready-made context comparator that is able to calculate the suitability value for PC and for the data objects that describe logical context expressions and Boolean, literal, numeric, time, and location context attribute elements. This context comparator is attached to these basic context elements. The named context elements can be attached to context-sensitive elements of a composition schema to calculate SFC values for PCs.

3. **Composition of task schemas.** A *task schema* is a data object that defines a task and its actions. A task schema composer takes the composition schema as an input and composes task schemas for various tasks. Finally, the task schemas are added to the *task factory* that is able to compose tasks for requests by using the task schemas as its base.
4. **Execution of a task.** The task factory composes a task for a request if a suitable task is not found from the task cache. It calls a thread to execute the task and its actions. Finally, the response of the task is delivered for the requestors.

Composition schemas define tasks and context-sensitive actions and their settings. Thus they will configure the components that are marked with a white colour in Figure 45.

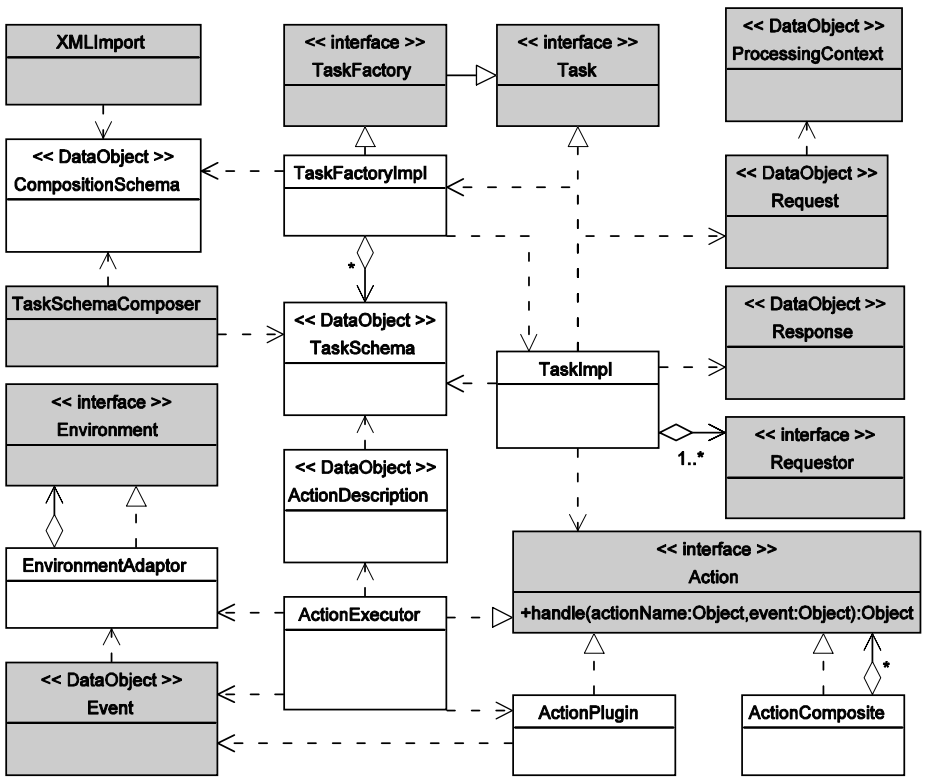


Figure 45. An implementation for the task-based composition technique.

TaskCAD offers implementations that are able to execute the action structures defined in composition schemas. In addition, it offers an *action executor* that calls the action plugins to execute the named actions of tasks (Figure 46).

An action is executed in the following steps. Firstly, it is checked that the action is suitable for the request and PC. If not, the action is not executed. Secondly, the settings elements are selected for PC and an environment adaptor is constructed for the defined Input and Output elements. Thirdly, the environment adaptor is passed in the event that the action plugin is called to perform a named action.

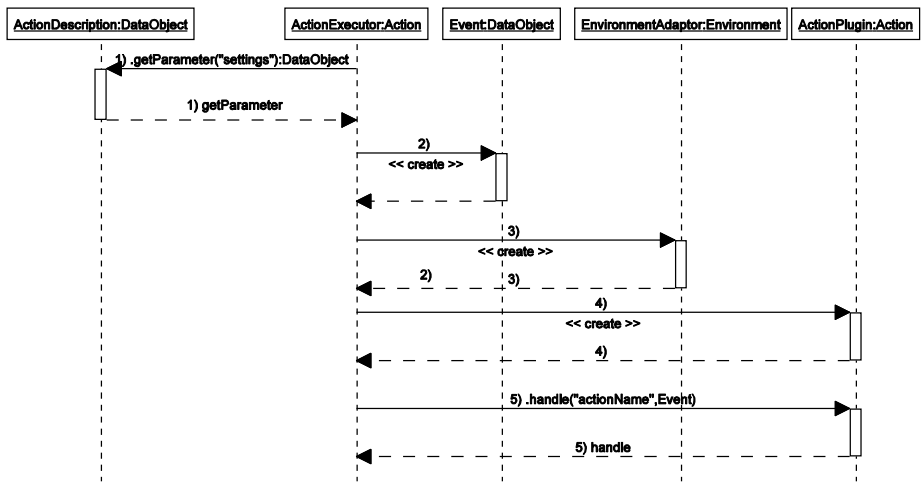


Figure 46. An action executor that calls the action plugin to perform a named action.

6.3 Execution structures for actions

The task-based composition language offers elements that can define various kinds of execution structures for adaptation actions. TaskCAD provides Java implementations for these elements. These are discussed in the following paragraphs.

The request action structure enables a task to request other tasks (Figure 47).

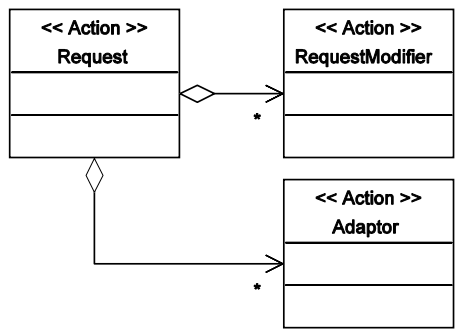


Figure 47. A request structure can have multiple actions modifying the request and adapting the received responses.

A request can contain actions that modify the request and adapt the received responses. A requestor can use an *action reference* and refer to the named action of the requested task and define settings for it in the request. The *action reference* is formatted as follows:

actionpath =actionname/[*actionpath*]
taskpath=taskname://[*actionpath*][*taskpath*]
actionreference=[*taskpath*]*actioname.*

For example, an action reference can have the form:

objectpull://openconnectionaction/requestplayer/prepareplayer://xhtmlplayer

As a result, the requestor can define settings for the referred *xhtmlplayer* action of the *prepareplayer* task that the *requestplayer* action of the *objectpull* task requests.

A request can also be cancelled. For example, if a refresh is made, a previously executed request is cancelled before a new request is made. A *respond* action delivers various kinds of responses for the requestors. If needed, it can also stop task execution after the response is delivered for the requestors.

A refresh action can contain actions that do post-processing related to a task and update its response (Figure 48). As a result, the requestors of the task are notified about the changed response. If the task is stopped, execution of refresh actions is stopped at the same time.

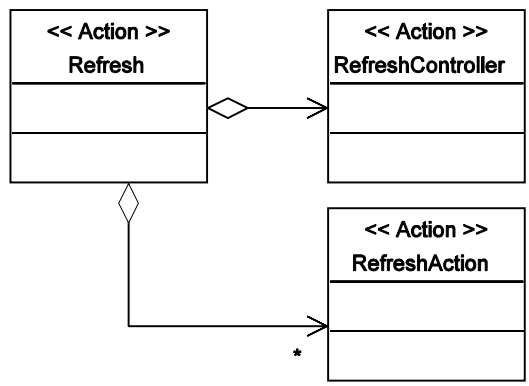


Figure 48. Refresh actions can do post-processing and update the response of the task.

An action sequence can contain multiple actions that are executed one after the other (Figure 49).

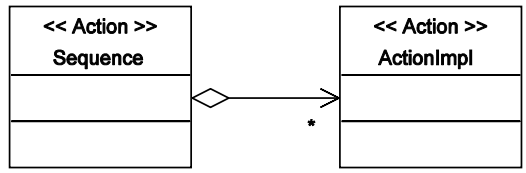


Figure 49. An implementation for an action sequence.

An exclusive action composite can contain multiple actions. It will execute only the first action that is suitable for PC or the action that is most suitable for PC (Figure 50).

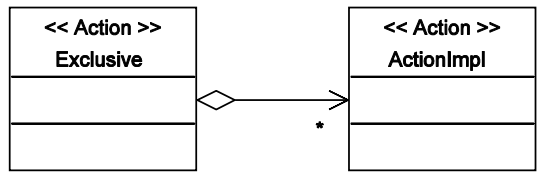


Figure 50. An implementation for an exclusive action composite.

An if-else action structure can contain conditional actions (Figure 51).

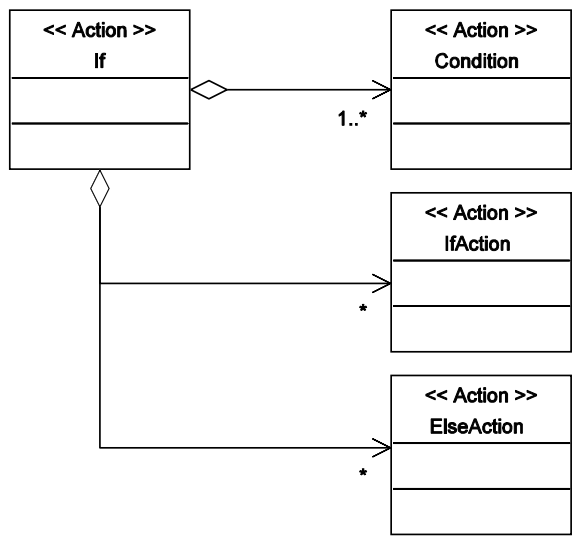


Figure 51. An implementation for an if-else action structure.

TaskCAD offers a ready-made condition that checks if an instance that is suitable for PC exists in the specified source. This can be used in an if-else structure that defines actions that need to be executed only if the instance does or does not exist in the specified source.

A *try-catch action structure* defines context-sensitive handling of exceptions that adaptation actions may throw (Figure 52).

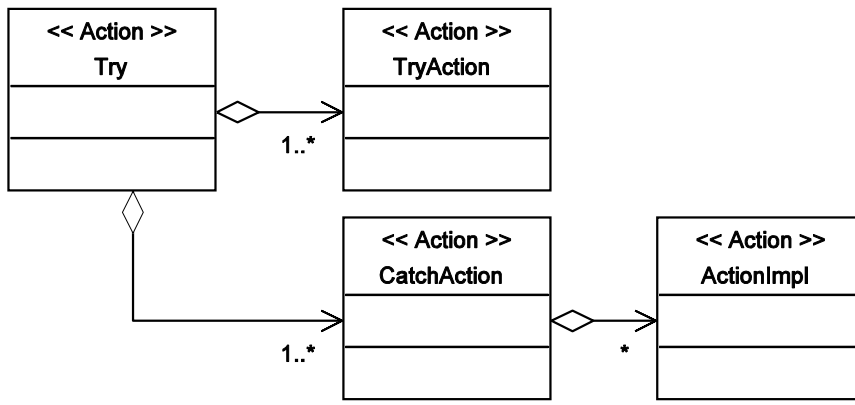


Figure 52. An implementation for a try-catch action composite.

The try-catch action composite throws an exception, if a suitable catch action composite is not found for PC. The action event offers a method that enables actions to stop execution of the task. The method throws a stop execution exception. The run method of the task catches the exception and finally stops execution of the task. It must be noted that the stop execution exception is a special exception, and thus, for example, the try-catch action structure does not handle this exception type.

An enumerate action structure repeats the defined actions for all the child objects of a named data object (Figure 53). The enumerated child object is stored to the execution environment of the task. As a result, the actions of an enumerate action can use the enumerated child object during execution.

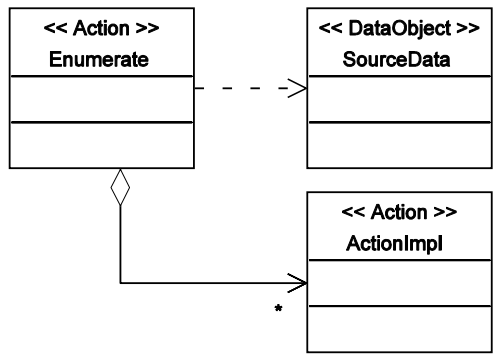


Figure 53. An implementation for an enumerate action.

An accept action structure enables the user to control the adaptation (Figure 54). Acceptor action can display user interfaces that enable the user to control the adaptation. At the same time, the execution thread of the accept action is set to the wait state. After the user has made a selection, the thread of the accept action is activated and either the accepted or rejected actions are executed.

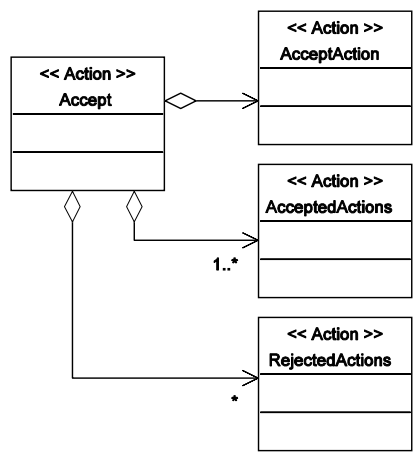


Figure 54. An accept action structure enables the user to accept, reject, or skip actions.

A select action structure enables the user to select between optional actions (Figure 55). The selector actions can present a user interface first enabling the end-user to select between alternative adaptation actions. At the same time, the thread of the select action is set to the wait state. It is activated after the end-user

selects an option. As a result, the actions that the selected option contains are executed. The default option is used if the user does not select any option during the defined time-out.

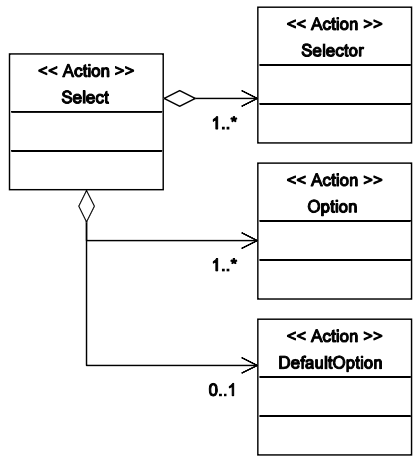


Figure 55. An implementation for the select action structure.

An ActionComposite structure can contain various kinds of actions and define a plugin that introduces a new execution structure for the defined actions.

6.4 An application environment and components for a task and application instance caching

TaskCAD offers ready-made implementations for the execution, cache, and application environments. The environments can have alternative instances prepared for various contexts. As a result, multiple application instances can be found for an Instance Reference (IRef). In this case, the most suitable one is selected for PC.

A context-sensitive application instance is stored to the environment as a data object that contains an original instance, context comparators (Figure 56), and PC for it.

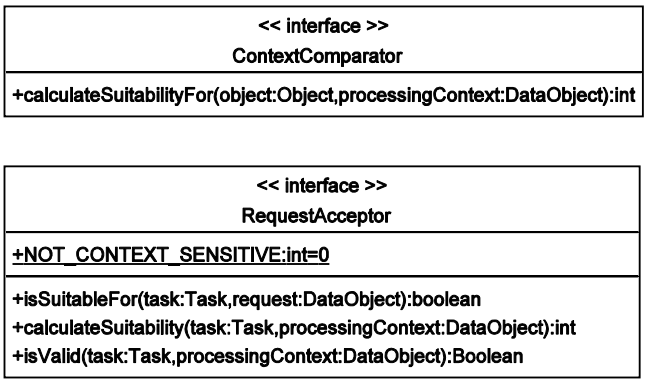


Figure 56. ContextComparator and RequestAcceptor interfaces.

TaskCAD offers ready-made context comparators for instances that are suitable for a specific PC only, for instances that are suitable for certain PC values only, and for instances that are suitable for all contexts (see Section 4.5.4).

RequestAcceptor provides three methods that are able to control caching of tasks (Figure 56). The first method defines if a task is suitable for a request whereas the second one calculates how suitable the task is for the processing context. The *isValid* method returns true if the task is suitable for given PC. TaskCAD offers ready-made *RequestAcceptors* (see Section 4.5.4) for tasks that do not use PC values, for tasks that utilise only part of the PC values, and for tasks that utilise all the PC values.

6.5 An XML editor for task-based composition schemas

It can be difficult to read and edit a composition schema if a lot of task, action, and context definitions are specified within it. A framework (called FEdXML [PaL05c]) facilitates implementation of component-based XML editors. We used FEdXML and implemented a specialised XML editor to help creation of composition schemas (Figure 57). It gives a visual presentation for the composition schemas and thus helps developers to see the tasks and actions of the composition schemas.

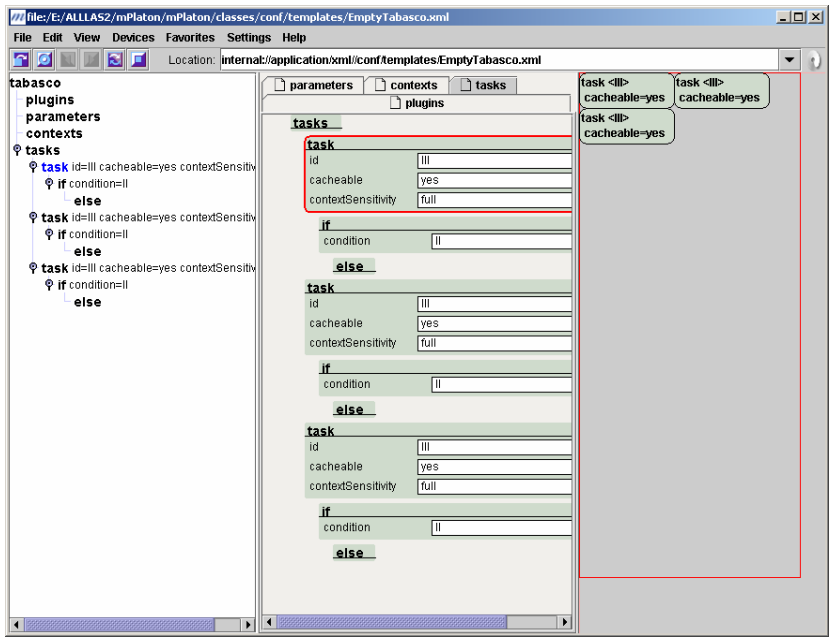


Figure 57. An XML editor for task-based composition schemas.

The editor offers basic editing methods (e.g. XML elements can be added, removed, or replaced) and undo and redo operations for XML editing. In addition, XML-based templates can be used in construction of composition schemas. For example, ready-made templates are provided for adaptation actions (e.g. for glyph composition actions). Furthermore, new templates can be added to the editor environment. For example, if the developer implements a new action, he or she can make a new template that defines an action element and default settings elements for the action and finally attach the template to the editor environment. As a result, it is faster to utilise the action plugin later in the composition schemas.

The editor also provides a tool that generates an empty action plugin for an action element defined in the composition schema. The Input and Output elements of the Action element define names for instances that the plugin may fetch from defined sources or add to specific targets. The tool generates an empty method to the plugin and then adds comment lines that contain a source code for these fetch and add operations. The developer can use a generated action plugin as a base and write code lines that will implement the actual functionality of the action.

7. Case studies – Utilizing the task-based composition technique for adaptive mobile browsers

The goal of the task-based composition technique is to make the dynamic composition of content and context-sensitive applications more fluent and to help developers to implement adaptive applications for mobile devices. The task-based composition technique supports dynamic composition of various kinds of component-based adaptive applications. This dissertation uses adaptive mobile browsers as the basis for a case study. The case studies discussed in Sections 7.3, 7.4, and 7.5 are based on the usage scenarios presented in Chapter 4. Case studies are described, implemented, and finally evaluated with measurements. Both performance and implementation benefits are discussed in subsections. Finally, summaries are provided for all the case studies.

7.1 Introduction

Mobile usage is spontaneous and requires applications that can fluently adapt for new contexts. The performance of adaptation must be optimized. At the same time, it is important that the implementation of an adaptive application for a mobile device does not require too much effort. TaskCAD has many features that promote implementation of applications that can fluently adapt for new contexts. Firstly, the used adaptation policies (composition schemas) and actions can be replaced with new ones at runtime. Secondly, it facilitates the caching of both adaptation tasks and application instances. Thirdly, it supports speculative adaptation, context-sensitive handling of errors, and utilisation of new execution structures in adaptation and provides structures that enable the user to control the adaptation.

Browsers are a very generic way to implement UIs for various kinds of Internet services. Specialised browsers can offer UIs for almost any kind of applications. In addition, they can be embedded in adaptive applications to enable the end-users to use Internet services directly in applications that are executed in a mobile device. The local network capabilities and services available for a nomadic user may change constantly. In addition, the heterogeneity of mobile

devices is high. A mobile browser should be highly configurable and it should be possible to configure and replace the components of the browser in order to adapt it for various contexts, wireless network connections, and for specific services available on the Web.

We implemented a framework, called MIMEFrame, to facilitate composition of component-based mobile user agents and browsers [PaL03, PaL06] and to provide ready-made interfaces and components for different kinds of user agent and browser implementations (Section 7.2). Tasks can compose content and context-sensitive mobile browsers of MIMEFrame components (Section 7.3), context-sensitive UIs for physical environments (Section 7.4), and improve the utilisation of the services of Bluetooth access points (Section 7.5).

The case studies discussed in Sections 7.3, 7.4, and 7.5 evaluate both the performance and implementation benefits of the task-based composition technique. The performance benefits are evaluated by measuring the execution times of the applications that were implemented in the case studies. The implementation effort is evaluated by measuring the implementation time and by calculating the total amounts of code lines that these application implementations required. However, it is important to notice that the given evaluations do not compare TaskCAD to other available adaptation methods but only show what kind of implementation and performance benefits it provided in the described adaptive browser implementations. Thus future research and experiments are still needed to provide numerical measurement information about alternative adaptation methods. For example, more information about the performance and implementation benefits of available adaptation methods is still needed.

7.2 A framework for adaptive browsers

In MIMEFrame, a browser is divided into *user agent* (model), *player* (view) and *controls* (controller) parts according to the *Model-View-Controller* pattern [KrP88] (Figure 58). The *user agent* is an abstract model for the client device and the user. The *player* performs the client-side adaptation and presents the browsed content. Finally, *controls* provide the way in which the browser is used.

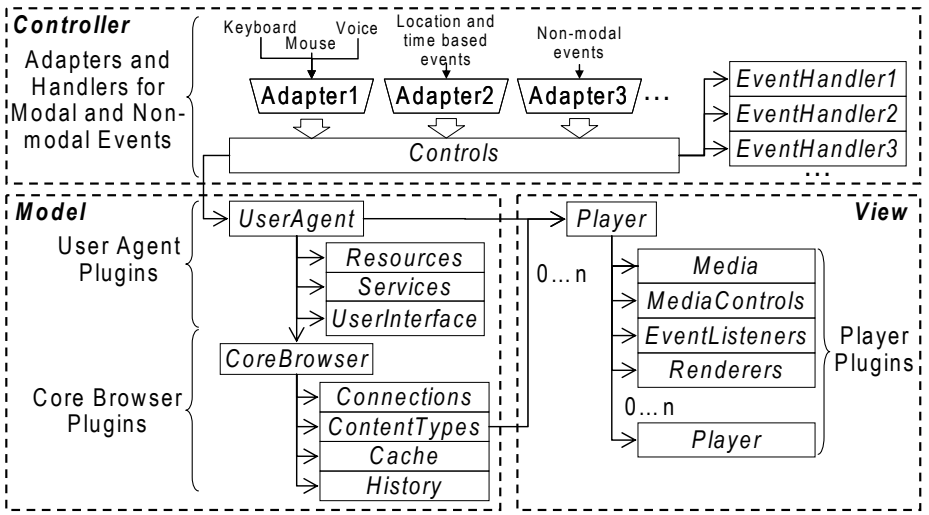


Figure 58. The basic parts of a browser.

The *UserAgent* offers core methods and various kinds of *resources*, *services* and *UIs* to be used in browsing. It can offer access to the methods of the actual UI component and methods that are capable of handling errors raised during browsing. The UserAgent's synchronous and asynchronous *services* can prepare *resources* (e.g. a delivery context) for browsing.

The *CoreBrowser* offers asynchronous content fetching and pull (the user initiated) and push (externally invoked) browsing methods. It can attach delivery context to the Web requests in order to enable server-side adaptation. It offers methods for both content caching and navigating in the browsing history.

Player corresponds to e.g. a text, image, audio, and video or the container (e.g. WML, XHTML, and SMIL) content type plugin of a traditional browser. It can perform the client-side adaptation, illustrate contents, and use other players in illustrations. A player follows the glyph protocol [CaL90]. It can *request* screen space and *render* the content in the *allocated space*. As a result, players can be formed into *composites* that create an overall presentation for contents. By selecting players based on the content and context it is possible to compose *content-driven* and *context-sensitive presentations*.

In MIMEFrame, the plugin approach is extended so that players can both offer and utilize resources and services. For example, an XHTML player can register *variables* to the resources and can so enable the forms on the XHTML page to be filled with dialogs illustrated with a VoiceXML player. A player can offer synthesized speech, audio icons, and screen-based modal output services and *media controls* for specific content presentations. Animated contents are presented by repainting rendering components with a specified frequency. If a player is no longer used, the related memory resources are freed with a close method.

A browser can be controlled with modal and non-modal events. Modal events (keyboard, touch screen or voice controller) are raised by the user whereas non-modal events occur indirectly, e.g. push and instant messaging services, time and location-based events, server-side, and other mobile users may possibly control browsing. An *event handler* can be registered to *Controls* either for all or named events only. *Adapters* can modify, combine, and pass modal and non-modal events to *Controls* that can put them into a queue in temporal order and later notify the event handlers about them. *Adapters* can also reject events.

The MIMEFrame reference Java implementation offers user agent, player, and controls interfaces and classes to be used in various kinds of browsers. Ready-made players can present text, image, WML, XHTML MP, SVG, VoiceXML, audio, and video contents. An object browser, called AGB [PaL03], implements the methods of the *CoreBrowser*.

7.3 Implementing a content and context-sensitive browser with tasks

7.3.1 Application

Browsers typically support user initiated browsing. For example, navigate a new page, navigate in history (backwards, forward), and abort the navigation commands are available in common Web browsers [LNR96]. User initiated browsing commands can be executed with an object pull task (Figure 59).

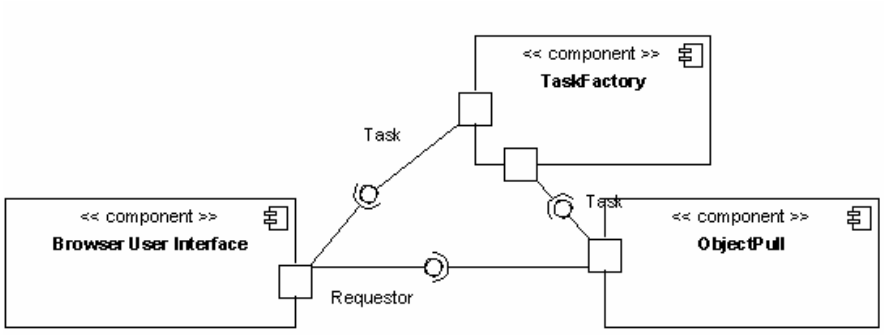


Figure 59. An object pull task can be requested to prepare a player for the contents available in the Web.

The browsing request defines source for the content to be browsed and may possibly define attributes for the network connections that are used in content fetches. An object pull task can fetch the content from the Web and prepare a player for it and finally deliver the player in its response. If errors are raised during execution, the object pull task delivers an error response that can be displayed for the user of the browser.

The browsed content can be *reloaded* in the following phases. A *reprocess* attribute can be set to the browsing request. As a result, the old object pull task is removed from the cache before a new object pull task is started for the request. If the player presents a container content type, the object pull task can be requested to prepare players for the referred contents (e.g. for the images of an XHTML page). The reprocess attribute can be set to these requests in order to ensure that the referred contents are reloaded, too.

The object pull task can have various kinds of actions that prepare a player for the browsed content in multiple phases (Figure 60). For example, it can have an action that requests a connector task to open a connection first. The adaptor action receives the response and can request a task to prepare a player for the content available via the connection.

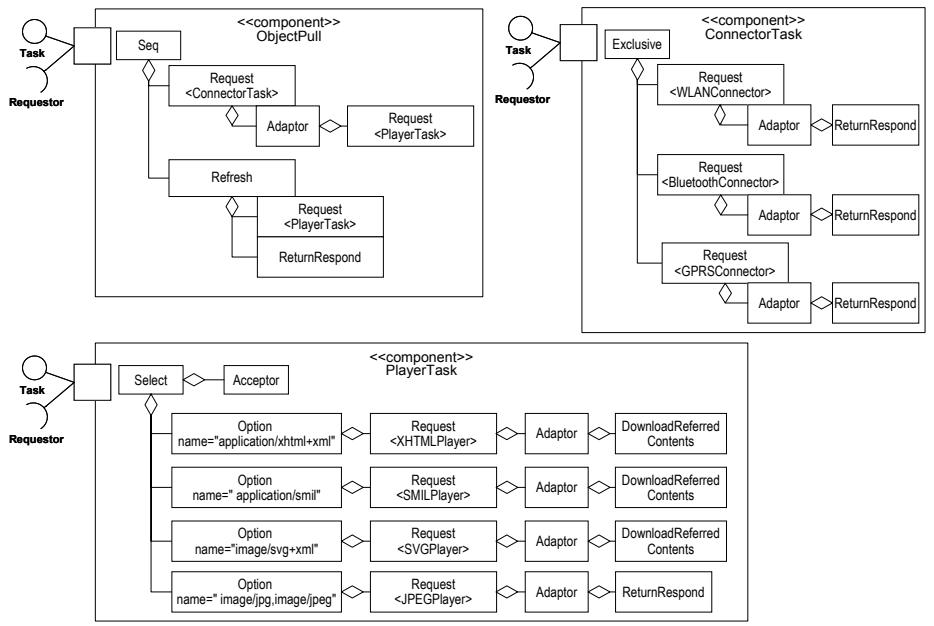


Figure 60. An object pull, connector, and player tasks.

The composition schema can define different tasks that can open various kinds of network connections (e.g. WLAN, Bluetooth, and GPRS). The connector task can request these network specific connection open tasks. Its request actions can be configured for various contexts so that only a task that will open the most appropriate network connection for PC is requested.

The player task can have a select action containing an acceptor plugin and options for various content types. The acceptor plugin can select an appropriate option for the content type. The request action of the selected option will request a task to prepare a player for the content type. If needed, the request action can have an adaptor action that requests object pull tasks to prepare players for the contents referred in the container (e.g. XHTML, SMIL, and SVG) content type. It receives the responses of the requested object pull tasks, composes the prepared (or updated) players to the player of the container content type, and notifies the requestors of the player task about the updated player.

The refresh actions can update a content presentation for the current PC in the following phases:

1. **The refresh actions fetch content elements for the PC.** If suitable content elements are not in the cache, refresh actions can reload the contents from the Web and store them in the cache. At the same time, the cached content elements that are not suitable for those current or possible forthcoming PCs can be removed from the cache. Context comparators are defined to control caching of the contents. For example, HTTP/1.1 offers a cache-control header to control caching both in proxy servers and clients [Luo98]. In addition, a Web server can specify an explicit freshness lifetime for the response by using the max-age directive. The context comparators can follow the directives of the cache-control header and also utilise the meta-information embedded in content elements and thus improve caching of context-sensitive content elements.
2. **Refresh actions compose new players for the contents and PC.** The players are delivered for the requestor of the object pull task that can update the browser view and finally display the new content presentation to the end-user.

It is important from the point of view of usability that the user gets all the time information about what the browser is currently doing. The *ObjectPull task* can provide the user with feedback with the info messages defined in the composition schema. For example, it can inform the user about opened connections and composed players.

If needed, a connection task can request an authenticate task that can have an accept action that stops execution of the task and displays authentication UI for the user (Figure 61).

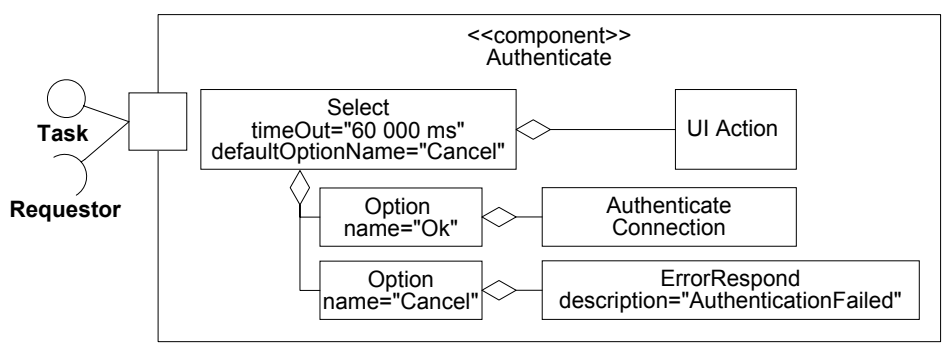


Figure 61. A task that is able to authenticate connections.

The user can input the required authentication information and accept the action. The task execution will continue and finally the authenticated connection is delivered for the requestors. If authentication is cancelled or failed, an error response will be passed for the requestors.

Section 4.9.1 proposes how tasks can compose content and context-sensitive UIs. TaskCAD enables developers to define specialised composition schemas for various kinds of content elements. Like a style sheet, a composition schema can describe how various content elements should be presented. However, style sheets typically only offer limited ways for specialising presentations. A composition schema can define tasks that are able to compose highly specialised and context-sensitive UIs for content elements. Both the functionality and appearance of UI can be adapted for the context.

The composition schema can be replaced dynamically with a new one that describes tasks that will compose UIs to better fulfil the user needs. In addition, it is easy to use tasks that can compose separate context-sensitive UIs for contents coming from separate sources and which finally compose these separate UIs to an overall UI. For example, actions can compose canteen menus only during the day to an overall UI whereas information about the movies and concerts can be displayed only on certain days of the week.

We implemented a browser prototype that can do client-side adaptation and which is capable of composing context-sensitive UIs for selected contents. Typically it is a lot faster to develop applications in a desktop computer than in a mobile device. Before an application can be executed in a mobile device, it must be compiled, transferred to, and started in the mobile device. This all requires time. Thus the first version of the browser prototype was implemented in the desktop environment with Java 2 Standard Edition (J2SE). The browser prototype was later ported to work in Java MIDP-enabled mobile devices.

The browser prototype was used to present context-sensitive bus timetables and restaurant menus. The browser displays a CSD list first and enables the user to select contents that he or she is interested in using. It requests tasks to fetch the selected contents and, finally, to compose an overall UI for them (Figure 62).

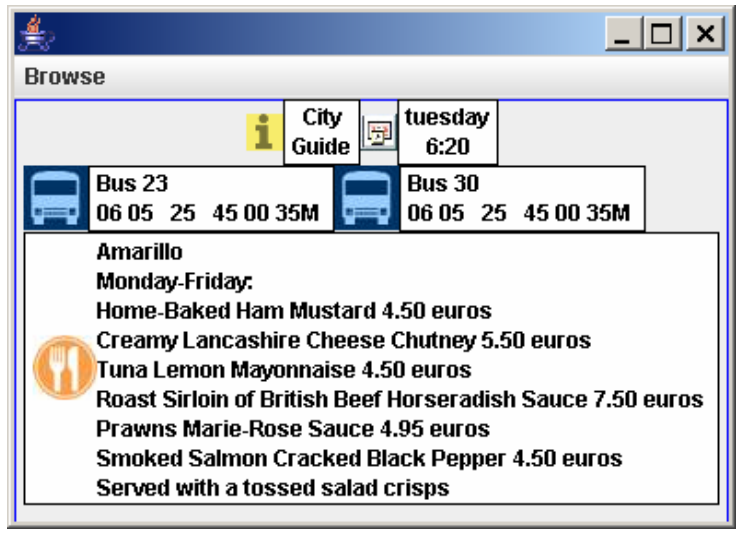


Figure 62. A browser implementation that composes an overall UI for the selected context-sensitive bus timetables and restaurant menus and displays it for the user.

The character, line break, line, polygon, and image glyphs and selectable, border, row, column, left-to-right, and table glyph composites of the MIMEFrame framework [PaL03, PaL06] were all utilised in the browser. The XML documents were parsed with a kXML 2 parser. The browser prototype was implemented in the following phases.

Firstly, XHTML documents were constructed for bus timetables and restaurant menus. The meta-information was attached to the content elements of these documents with span elements as presented in Section 4.9.1. Secondly, we constructed composition schemas that define tasks to compose a frame glyph for the overall UI and presentation glyphs for bus timetables, restaurant menus, and time.

Thirdly, we implemented actions to compose glyph-based UIs for content elements. The *PCModifier* action was implemented to add time and location context attributes to PC before the overall UI task was requested. The *XMLImport* action was used to parse XML-based composition schemas and content documents. It calls a *PlatformService* class to open a connection to defined Uniform Resource Locator (URL), fetches the XML document source then from

the Web, and finally parses a DataObject presentation for it. The *TaskFactoryImpl* action is called to initialise task factories for the fetched composition schemas. The *XHTMLMetaToolkit* is able to attach context comparators to the context-sensitive content elements. As a result, the correct content elements can be selected when a content presentation is composed for PC.

The content elements of composition schemas can be configured for various contexts. For example, they can describe glyphs and their attributes, composition conditions, and refer to (e.g. text and image) other content sources. The composition schema can attach these content elements to actions that are able to compose glyphs for these descriptions. For example, the frame glyph of overall UI was described in the composition schema. Furthermore, we implemented a *GlyphDescriptionComposer* action to compose glyph descriptions for content elements. It selects content elements for PC, composes a glyph description for the selected content elements, and finally adds it to the defined target (e.g. to the response or to execution environment of the task). The *GlyphComposer* action can be later called to compose presentation glyphs for the glyph descriptions, to add new glyphs to glyph composites, to remove named glyphs from frame glyphs, and to replace the named glyphs of frame glyphs with new ones. Finally, the *UIComposer* action composes a glyph-based UI to the browser view that displays it for the user.

The browser prototype was finally modified to work in Java MIDP-enabled mobile devices (Figure 63).

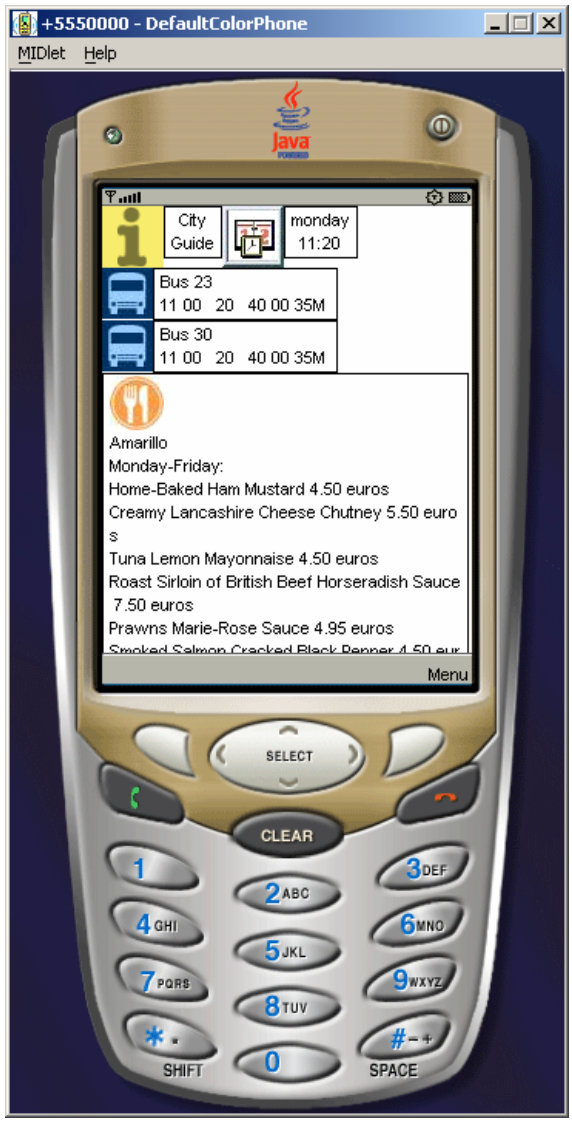


Figure 63. A Java MIDP implementation for the browser prototype.

7.3.2 Experiment setup

We made an experiment setup in order to evaluate task-based composition of content and context-sensitive browser-based UIs. We tested the browser prototype in a Nokia 6630 mobile phone and in a Universal Mobile Telecommunications System (UMTS) network. The presented execution times

were measured from the system clock. However, it must be noted that the data transmission speed can fluctuate in the UMTS network and thus affect the measurement results. In addition, the background processes that are running in the mobile device may also affect the execution times of the processing threads. We repeated the measurements three times in order to minimise the measurement errors. Each presented measurement result is the average of the three separate measurement values. A thread pool that had 100 Java threads was used in the measurements.

7.3.3 Performance benefits

A standard mobile browser and Web pages

The simplest way to browse bus timetables and restaurant menus is just to browse this information from separate Web pages. For example, it took a total of 11.5 seconds to fetch bus 23 and 30 timetables and restaurant menu and to display these separate documents with a Series 60 browser available in the Nokia 6630 mobile phone. However, it was difficult to read timetable information from the small display. In addition, it requires effort to browse separate Web pages.

Server-side adaptation

The server-side can compose overall context-sensitive presentation for content elements and thus improve performance and usability. The client can pass a CSD list and PC attributes in the request. The server-side can select and compose correct content elements to a single XHTML document and deliver it for the client that can finally display it for the user. For example, it took 4.3 seconds to fetch and display an XHTML document that contains only the selected content elements of bus timetables and restaurant menus. As can be seen, in this case it was faster to browse a single XHTML page when it was adapted for the context on the server-side. However, if the context changes, a new network request has to be made that requests the server-side to compose a new Web page for PC and to deliver it to the client. This causes delays and network traffic. Client-side content caching can decrease these delays. The client does not need to reload a Web page if it already has a Web page in its cache that is suitable for the new context.

Client-side task-based adaptation

Client-side adaptation can improve performance and usability and decrease network traffic. We attached meta-information (e.g. span elements) to bus 23 and 30 timetables and restaurant menu documents in order to enable the client-side to compose adapted content presentations for various contexts. The original XHTML documents took a total of 5 Kbytes. Meta-information increased the total size of these documents to 10.6 Kbytes. Thus the meta-information increased the total size of the XHTML documents by 112 percent in this example.

The content documents and composition schemas were fetched from the Web (Figure 64).

XHTML Document	Orginal Size [Kbytes]	Size with metainformation [Kbytes]	Size Increase	Download Time [s]	XML Parsing Time [s]
Bus 23 timetable	1,9	4,5	137 %	1,4	0,2
Bus 30 timetable	1,9	4,5	137 %	1,4	0,2
Restaurant Menu	1,2	1,6	33 %	0,8	0,1
Total	5 Kbytes	10,6 Kbytes	112 %	3,6 seconds	0,5 s

Figure 64. The browsed XHTML documents.

It took 3.6 seconds to download the XHTML document sources over an UMTS connection and a total of 0.5 seconds to parse them. The fetching and parsing of composition schemas took a total of 7.7 seconds. Finally, it took 6.7 seconds to compose context-sensitive presentation glyphs for the selected content elements. Thus composition of a context-sensitive UI for content elements selected for PC took total 18.5 seconds.

The parsed XHTML documents can be stored in the cache. The downloaded XHTML document was suitable for all PCs. Thus a context comparator that defines that an instance is suitable for all PCs was set to control caching of the XHTML document. It is possible to implement new context comparators that are able to identify more precisely for which kinds PCs a certain application instance is suitable or when it has expired and must be removed from the cache. For example, a context comparator implementation can utilise the meta-

information embedded in XHTML documents and thus improve the caching of XHTML documents.

The cached task factories can also be utilised when UIs are composed for selected content elements. This will speed up the dynamic composition of context-sensitive presentations. For example, it took 1.9 seconds to compose a new context-sensitive presentation for the cached bus timetable and restaurant menu documents.

For example, if the mobile device supports the push mechanism [Ort03], an inbound network connection or a timer-based alarm can wake a Java MIDlet up, configure it, and afterwards possibly close it if it seems that the user does not need the application anymore. One of the common presentation styles of push-type systems is a screen saver that enables the user to see information without performing any operations [SKSK98]. For example, a timer can wake a browser up to display bus timetables and restaurant menus at a certain time of the day. Afterwards, it can close the browser if it seems that the user does not need it anymore.

However, it must be noted that the user may incur a lot of unnecessary network traffic and costs if a browser is used as a screen saver. The browser is running continuously and so it can fetch a lot of contents from the Web. Caching can reduce the usage of the wireless network. For example, the implemented browser prototype does not make Web requests after updated bus timetables and restaurant menus are downloaded to the local cache but will use the wireless network only if the cached documents have expired.

7.3.4 Implementation benefits

The browser prototype required changes both on the client and server-side. The server-side must provide documents that contain content elements for various contexts. Furthermore, composition schemas for glyph-based UIs must be provided. Equally, an implementation for the task-based composition technique and actions that can perform the composition assignments defined in the composition schemas must be installed to the client. The implementation of these actions required coding effort but it may be possible to reuse these action

implementations in the future. In addition, composition schemas can be reused in new browser implementations, too.

We implemented the composition schemas of the browser prototype with the mPlaton editor (see Section 6.5). The total size of the composition schemas was 21.4 Kbytes. The sizes of these composition schemas are listed in the following table (Figure 65).

Composition Schema	Description	Size [Kbytes]	Download Time [s]	XML Parsing Time [s]
RootTask	Attaches time and location attributes to PC before requests the OverallPresentation task.	5,2	1,7	0,1
OverallPresentation	Fetches contents and requests other tasks to compose a presentation frame and presentations for downloaded contents.	6,7	1,8	0,1
PresentationFrame	Composes a glyph frame for the overall presentation.	3,0	1	0,1
BusSchema	Composes a glyph presentation for a bus timetable.	2,7	0,9	0,1
RestaurantSchema	Composes a glyph presentation for a restaurant menu.	2,7	0,9	0,1
TimePresentation	Composes a glyph presentation for time.	1,1	0,8	0,1
	Total	21,4 Kbytes	7,1 s	0,6 s

Figure 65. The composition schemas used in the browser prototype.

The mPlaton editor facilitated implementation of composition schemas. It took only 4 hours to implement the composition schemas.

We implemented an abstract *PlatformService* class and platform (J2SE and Java MIPD) specific implementations for it to provide methods for connection opening, class loading, and for floating point calculations. Furthermore, the PlatformService class offers a service that composes the browser view (a frame in J2SE and a canvas in Java MIDP environment) that displays the composed

glyph-based UIs for the user. Thanks to the PlatformService class, all the action implementations worked directly both in the J2SE and in the Java MIDP environments. At the same time, it was much easier to update the browser prototype because the same actions and composition schemas are used in both environments.

The actions required total 779 lines of coding. Furthermore, classes for the platform specific services required a total 492 of lines of coding (Figure 66).

Action	Description	Size [lines of code]
UIComposer	An action that calls the PlatformService class to compose the browser view that can display glyph-based UIs.	11
PCModifier	Adds time and location information to PC.	78
XHTMLMetaToolkit	Offers methods that are able to parse the content attribute value and compose context elements that define acceptable PCs for a particular content element.	220
GlyphDescriptionComposer	Selects content elements for PC and composes glyph descriptions for the selected content elements. In addition, it is capable of composing a glyph description for time.	145
GlyphComposer	Composes presentation glyphs for a glyph description.	325
	Total	779 lines

Platform Specific Implementation	Description	Size [lines of code]
J2SEPlatformService	An implementation for the PlatformService class in the J2SE environment.	208
JavaMIDPPlatformService	An implementation for the PlatformService class in the Java MIDP environment.	284
	Total	492 lines

Reference Implementation	Description	Size [lines of code]
XMLImport	Opens a connection to defined URI, parses the fetched XML document, and finally composes a data object presentation for it.	145
TaskFactoryImpl	An action that is able to initialise a task factory.	35
Sequence, Request, Enumerate, Refresh, and If-Else Actions	These action execution structures were used in the composition schemas.	576
ContextElementComparator	This component is used when the content elements are selected for PC.	165
	Total	921 lines

Figure 66. The action implementations used in the browser prototype.

TaskCAD provides execution structures for adaptation actions. The *sequence*, *request*, *enumerate*, *refresh*, and *if-else* execution structures of TaskCAD were

used in the browser prototype. Thus a total of 921 lines of code of TaskCAD were used in composition of the browser prototype.

7.3.5 Summary

The size of the browser installation package was 126 Kbytes. Mobile usage is spontaneous and for this reason it is important that applications are fast to use. It took 4.2 seconds to start the task-based browser.

Figure 67 presents delays for alternative client and server-side content delivery and presentation approaches. The delay was 11.5 seconds when the different documents were downloaded with a standard browser. The server-side can compose an overall context-sensitive presentation for content elements. It took 4.3 seconds to download and display an XHTML document that contained only the selected content elements of bus timetables and restaurant menus. Tasks were used in the client-side adaptation. It took 18.5 seconds to compose a presentation on the client-side when the composition schemas and content elements were downloaded from the Web. It took 6.0 seconds to download contents and compose a presentation for them when the cached task factories were utilised. Finally, it took 1.9 seconds to compose a context-sensitive presentation when the cached contents and task factories were utilised in dynamic composition.

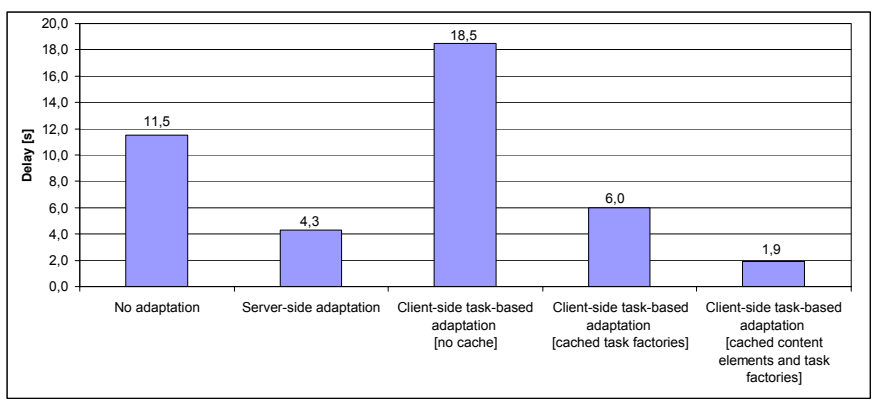


Figure 67. Delays for alternative content delivery and presentation approaches.

In this example the task-based composition technique reduced the total coding effort by 48 percent in J2SE and by 46 percent in Java MIDP environment. However, it must be noted that the implementation of the XML-based composition schemas requires effort. It took only about 4 hours to implement the composition schemas for the browser prototype with the mPlaton editor.

Although MIMEFrame and TaskCAD facilitate implementation of adaptive browsers, it may be too difficult task for developers of limited programming experience to construct totally new adaptive browsers. However, if a lot of ready-made browser components and composition schemas are available, it is possible to implement new kinds of browsers without manual coding. Only composition schemas have to be edited. This can be made with standard text or XML editors. In addition, the mPlaton editor facilitates implementation of the composition schemas.

7.4 Task-based composition of UIs of physical environments

7.4.1 Application

The physical environment can offer local services for nomadic users. For example, a home automation system can provide various UIs for different locations at home that enable the user to observe and control home appliances. A push message can start the application, which can in turn compose context-sensitive UI for the objects of the physical space in which the user is located. These UIs can be composed with tasks as presented in Section 4.9.2.

A physical space and its objects may change constantly. This requires dynamically changing UIs that are adapted for the physical space in question. Unlike many mark-up languages, a composition schema does not necessarily directly define UI or content elements but describes how UI should be composed for physical spaces, objects, or contents in various contexts. For example, if a room has five light switches, UIs of different switches can be composed on the client-side by using a single composition schema and without downloading separate UIs for various light switches.

The task-based composition technique facilitates the composition of dynamic UIs of physical spaces and supports client and server-side adaptation. Tasks can compose UIs on the client-side and so it is possible to reduce the amount of network traffic when UIs are adapted for new contexts.

If needed, the XML-based composition schemas can be modified or replaced with new ones. For example, the user may fetch a new composition schema from the Web that can compose better UIs for certain kinds of physical spaces. Furthermore, it can introduce a new layout and overall UI for physical spaces or replace the default UIs of the physical objects with new ones.

A composition schema can define very fine-grained adaptations to adapt both the visual appearance and functionality of glyph-based UIs. For example, context-sensitive (e.g. font, color, and position) parameters can be defined for glyphs. Parameters can also define what kind of functionality is invoked when the user clicks a selectable glyph in the UI. Of course, this requires a plugin that is capable of executing the defined functionality. The content elements and attributes can be configured for various contexts and attached to actions that can compose context-sensitive and glyph-based UIs for content elements. For example, the light switches can be hidden in the UI in the day time. An action can also compose glyph presentations for the data objects referred to in the composition schema. For example, it can compose character glyphs for a light switch name (Figure 68).

Actions can also use the state attributes of physical objects in composition. For example, an action can compose a glyph for a defined icon to display the status of the light (on or off) and add it to the glyph presentation. The “light status” glyph can be a *selectable glyph* that enables the user to select or click the glyph and to invoke the specified functionality. For example, the “light status” glyph can invoke an action that will send a request to an ObjectURI that is defined in a Physical Object Description (POD). The server-side can handle the request and turn the light on or off.

```
<context id="daytime">          Contexts can be defined in
                                the composition schema.
    <and weight="1">
        <number type="time" relationop="GreaterThan" value="10:00" weight="1"/>
        <number type="time" relationop="LowerThan" value="18:00" weight="1"/>
    </and>
</context>                       A light switch is not visualised at daytime.
...                              The context "daytime" is described in the
<content name="GlyphDescription">  composition schema.
    <cattr name="isVisible" value="false" context="daytime"/>
    <cattr name="glyph" value="LeftToRightComposite"/>
    <content>                                  The POD is delivered in the
        <cattr name="glyph" value="ImageGlyph"/>  request. It defines a name for
        <cattr name="src" value="/LampOnIcon.jpg"/>  the light switch. Character
        <cattr name="width" value="20"/>          glyphs are composed for it
        <cattr name="height" value="20"/>         and added to the left-to-right
        <content>                                 glyph composite.
            <cattr name="Variable" value="request#POD.name"/>
            <cattr name="glyph" value="CharacterGlyph"/>
        </content>
    <content>      The settings define input and output data sources for the action that
</content>         composes glyphs for the physical object and glyph descriptions.
...
<action name="composeGlyphFrame" plugin="fi.vtt.tte.ccpresentation.GlyphComposer">
    <settings>
        <input name="frameName" sourcePath="request#POD.type"/>
        <input name="frameURI"  sourcePath="request#POD.objectURI"/>
        <input fetch="fetch_best_for_IRef_and_PC"          Reference to content
               name="glyphdefinition"                     elements that
               sourcePath="settings#content#GlyphDescription"/>  describe visualisation
        <output insertStyle="replace" name="glyphframe" targetPath="response#podui"/>  elements for a light
    </settings>                                                                        switch.
</action>
...
```

Figure 68. A composition schema can define context-sensitive glyphs for Physical Object Descriptions (PODs).

We implemented a task-based browser prototype that displays context-sensitive UIs for the user moving at home and enables the user to control and observe home appliances through nearby Bluetooth access points (Figure 69).

Figure 69. A task-based browser prototype that composes context-sensitive UI for the objects of a physical space and displays it for the user.

The desktop version of the browser prototype was first implemented with Java 2 Standard Edition (J2SE). The character, line break, line, polygon, and image glyphs and selectable, border, row, column, left-to-right, and table glyph composites of the MIMEFrame framework were utilised in the browser. The XML documents were parsed with a kXML 2 parser. The browser prototype was implemented in the following phases.

Composition schemas were first implemented for overall UI and for various kinds of physical object types (e.g. for TV, lights, time, and temperature). Then, we implemented the actions that were used in composition schemas. A part of the implementations used in the content and context-sensitive browser prototype (see Section 7.3) was reused in the POD browser. For example, *PCModifier*, *XMLImport*, *GlyphComposer*, and *UIComposer* actions and the *PlatformService* class were utilised in it.

The browser prototype was finally modified to work in Java MIDP enabled mobile devices (Figure 70).

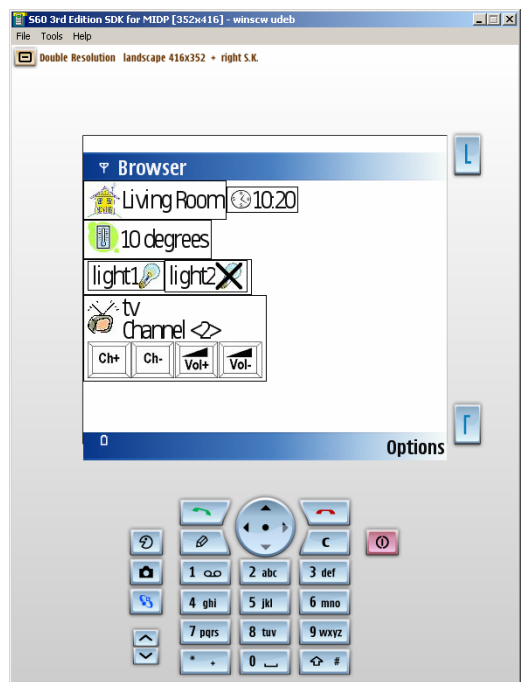


Figure 70. A Java MIDP implementation for the browser prototype.

Thanks to the PlatformService class, all the action implementations worked directly both in the J2SE and in the Java MIDP environments.

Finally, we implemented composition schemas that defined tasks to compose PODs for nearby physical objects. The J2SE version of the browser had local composition schemas for the PODs tasks whereas the Java MIDP browser acquired the composition schemas of PODs tasks from Bluetooth access points.

7.4.2 Experiment setup

We made an experiment setup in order to evaluate task-based dynamic composition of content and context-sensitive browser-based UIs. We tested the browser prototype in a Nokia N70 mobile phone and in UMTS and Bluetooth networks. The presented execution times were measured from the system clock. However, it must be noted that the connection opening time and data transmission speed can fluctuate in the Bluetooth and UMTS networks and thus affect the measurement results. In addition, the background processes that are running on the mobile device may also affect the execution times of the processing threads. We repeated the measurements three times in order to minimise measurement errors. Each presented measurement result is the average of the three separate measurement values. A thread pool that had 100 Java threads was used in the measurements.

7.4.3 Performance benefits

The browser installation package took 123 Kbytes. Mobile usage is spontaneous and so it is important that applications are fast to use. It took 4.5 seconds to start the task-based browser. The content elements and composition schemas were fetched from the Web. In addition, the following images were used in the composed UI (Figure 71).

Image	Size [Kbytes]	Download Time [s]
TVIcon.jpg	1,4	1,1
TV_CHMIN_Icon.jpg	1,4	1
TV_CHPLUS_Icon.jpg	1,5	1,1
TV_VOLMIN_Icon.jpg	1,6	1,1
TV_VOLPLUS_Icon.jpg	1,6	1,1
HouseIcon.jpg	1,4	1,2
LampOffIcon.jpg	1,2	1,1
LampOnIcon.jpg	1,0	1,1
TimeIcon.jpg	1,0	1
TemperatureIcon.jpg	1,2	1,1
Total	13,4 Kbytes	10,9 seconds

Figure 71. The images that were downloaded from the Web.

Client-side adaptation

The average time to scan for services available via Bluetooth access points was 18.1 seconds in the Nokia N70 device. In addition, fetching a composition schema for a PODs task (size was 7.2 Kbytes) and initialising a task factory for it took total 4.8 seconds. The fetching and parsing of composition schemas took total 8.9 seconds. Finally, it took 10.9 seconds to fetch content elements (images) and 7.8 seconds to compose context-sensitive UI for PODs. Thus composition of context-sensitive UI for PODs took a total of 50.5 seconds.

Caching of task factories and application instaces

In many cases this delay is far too long time for the spontaneous mobile usage. Thus it is important to shorten this delay. Firstly, the time to scan for Bluetooth access points took a lot of time. A speculative adaptation can decrease this delay (see Section 7.5). Secondly, the caching of composition schemas and contents can shorten this delay a lot. In the best case, if the needed resources are in the cache, a UI can be composed for the new PC without causing network traffic at all. Only, updated PODs need to be acquired. This can speed up adaptation. For example, it took 21 seconds to scan for Bluetooth APs, fetch PODs from an access point, and compose a UI for PODs when the cached task factories and

content elements were utilised in dynamic composition. However, in many cases this delay is far too long for spontaneous mobile usage.

Bluetooth AP scanning is unnecessary and a lot of time is saved if the addresses of the nearby BT access points are made available in the client cache. For example, acquiring a new PODs composition schema from a known Bluetooth access point and composing the UI for it took only 2.9 seconds when the task factories and images were stored in the local cache.

Speculative adaptation

Speculative adaptation tasks can be used to shorten composition delays. For example, if the route of the user can be predicted, UIs for the locations to which the user will possibly go next can be composed in the background. Speculative adaptation tasks can fetch composition schemas, initialise task factories for these, and finally request their tasks to compose UIs for the next locations in the background. The initialised task factories and composed UIs can be stored in the cache. As a result, these UIs can be fast displayed for the user when he or she arrives at a specific physical space. In addition, it may be possible to utilise the cached task factories in the future when UIs are composed for forthcoming PCs.

7.4.4 Implementation benefits

Task-based composition of UIs of physical spaces requires changes both on the client and server-side. The server-side must provide physical object descriptions for objects that can be controlled through the available servers. The *BluetoothConnector* framework facilitated the implementation of the POD server. It required only 300 lines of coding to implement a server that delivers a composition schema that contains randomly generated physical object descriptions. However, it must be noted that the described implementations are prototypes only and they are not integrated into a real ubiquitous environment. More effort is needed, if the server-side integration is desired in a real ubiquitous environment.

Tasks to compose glyph-based UIs for physical spaces and for various kinds of physical objects are needed, too. Finally, an implementation for the task-based

composition technique and actions that can perform the composition assignments defined in the composition schemas must be installed to the client.

It took about 3 hours to implement the composition schemas for the browser prototype with the mPlaton editor. The same composition schemas were used in the J2SE and Java MIDP browser implementations. The total size of the used composition schemas was 24.8 Kbytes (Figure 72).

Composition Schema	Description	Size [Kbytes]	Download Time [s]	XML Parsing Time [s]
PUICaseStudy.xml	Attaches time and location attributes to PC before requests the PUI task.	6,2	1,8	0,1
PUI.xml	Requests other tasks to compose a glyph frame for overall UI and glyph-based UIs for PODs.	4,5	1,5	0,1
PUIFrame.xml	Composes a glyph frame for the physical user interface.	3,3	1,1	0,1
locationschema.xml	Composes a glyph presentation for a location information.	1,7	0.9	0,1
timeschema.xml	Composes a glyph presentation for time.	2,0	0,9	0,1
temperatureschema.xml	Composes a glyph presentation for temperature.	1,8	0,9	0,1
tvschema.xml	Composes a glyph presentation for the remote control of TV.	3,1	1,0	0,1
lightschema.xml	Composes a glyph presentation for light switches.	2,3	0,9	0,1
	Total	24,8 Kbytes	8,1 seconds	0,8 seconds

Figure 72. The composition schemas used in the browser prototype.

The task-based composition of UIs of physical spaces requires actions that can acquire PODs from available access points, fetch the necessary composition schemas from the Web, and finally compose glyphs for overall UI and for found PODs. The *BluetoothConnector* framework facilitated utilisation of Bluetooth connections. Only the methods that call BluetoothConnector to scan for the available Bluetooth access points, to open connections to the access points that provide the POD service, and to acquire PODs were implemented. In addition, plugins executing the functionalities related to selectable glyphs were

implemented. The actions implementations required total 516 lines of coding. Furthermore, classes for platform specific services required a total of 696 lines of coding (Figure 73).

Action	Description	Size [lines of code]
UIComposer	An action that calls the PlatformService class to compose the browser view that can display glyph-based UIs.	11
PCModifier	Adds time and location information to PC.	78
PODFunctionality	The POD functionality that is attached to a selectable glyph will send a request for the Bluetooth server that is controlling the physical object in question.	102
GlyphComposer	Composes presentation glyphs for a glyph description.	325
	Total	516 lines

Platform Specific Implementation	Description	Size [lines of code]
J2SEPlatformService	An implementation for the PlatformService class in the J2SE environment.	208
JavaMIDPPlatformService	An implementation for the PlatformService class in the Java MIDP environment.	284
BluetoothPODService	An implementation that fetches composition schemas from Bluetooth access points in the Java MIDP environment.	204
	Total	696 lines

Reference Implementation	Description	Size [lines of code]
XMLImport	Opens a connection to defined URI, parses the fetched XML document, and finally composes a data object presentation for it.	145
TaskFactoryImpl	An action that is able initialise a task factory.	35
Sequence, Request, Enumerate, Refresh, and If-Else Actions	These action execution structures were used in the composition schemas.	576
ContextElementComparator	This component is used when the content elements are selected for PC.	165
	Total	921 lines

Figure 73. The action implementations used in the browser prototype.

The *sequence*, *request*, *enumerate*, *refresh*, and *if-else* execution structures of TaskCAD were used in the browser prototype. Thus total 921 lines of code of TaskCAD were used in composition of the browser prototype.

7.4.5 Summary

The delays for UI composition tasks are presented in Figure 74.

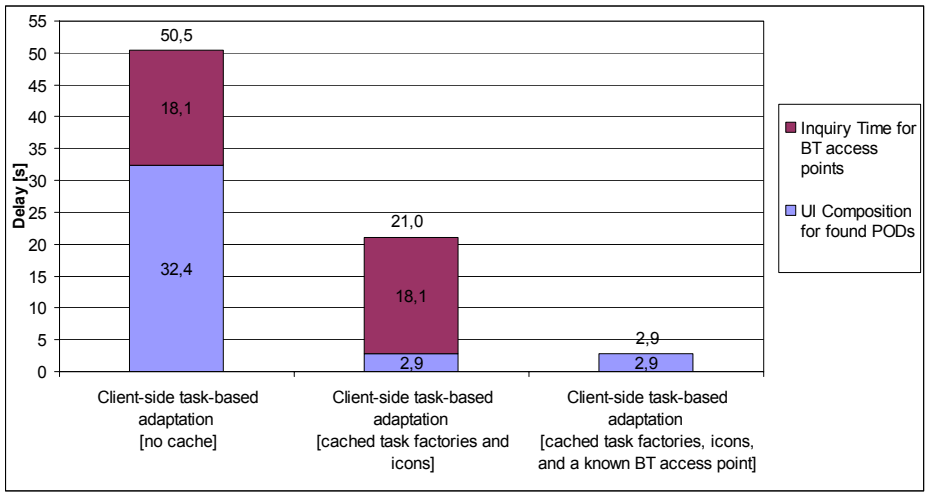


Figure 74. Caching of task factories and content elements made it faster to compose UIs for PODs.

Firstly, it took a total of 50.5 seconds to scan for Bluetooth APs, fetch PODs from an access point, and compose a UI for PODs when the required task factories and contents were not available in the local cache. Secondly, it took 21 seconds to scan for Bluetooth APs, fetch PODs from an access point, and compose a UI for PODs when the cached task factories and content elements were utilised in dynamic composition. Thirdly, it took only 2.9 seconds to acquire a new PODs composition schema from a known Bluetooth access point and to compose a UI for it when the cached task factories and required contents were stored to the local cache. As can be seen, the caching can significantly decrease the composition delay. However, it takes a lot of time (over 15 seconds) to scan for available BT access points. Speculative adaptation methods are needed to reduce the disconnection time in Bluetooth environments.

In this example the task-based composition technique reduced the total coding effort by 56 percent in J2SE and by 47.8 percent in Java MIDP environment. In

addition, it took about 3 hours to implement the composition schemas for the browser prototype with the mPlaton editor.

7.5 Using speculative adaptation tasks to shorten the disconnection time in browsing of local services

7.5.1 Application

In order to make use of local services fluent, there should be seamless connectivity at network. However, it takes time to open connections to access points. As a result, the mobile device may be disconnected from the network when the user moves in a local service environment or between various service environments.

For example, Bluetooth (BT) access points (APs) can provide local services for mobile users. Connections are typically opened to BT access points in the following phases. Firstly, available BT access points are scanned. Secondly, a connection is opened to an access point that provides the needed service. The AP inquiry time is typically more than ten seconds (e.g. 18.1 seconds in a Nokia N70 device) and thus significantly increases the disconnection time. Speculative adaptation can shorten the disconnection time when the user is moving in a specific physical environment (e.g. inside a building). If the addresses of APs are known, the AP inquiry is not needed but direct connection attempts can be made to known APs that are available in the building.

The task-based composition technique supports speculative adaptation. Composition schemas can be fetched from the Web, modified, or replaced with new ones. Thus it is possible to change adaptation strategies at runtime. For example, new composition schemas for speculative connection open tasks can be fetched to optimise the usage of local services and to shorten the disconnection time when the user is moving in a specific physical space (e.g. inside a building). They can define the addresses for local access points and parameters that may configure the prediction models and thus improve the prediction accuracy.

For example, if the route of the user can be predicted, an attempt can be made to connect to access points that are available at the next locations. The floor plan of

a building can set limitations on the routes available to the user. This information can be delivered in composition schemas and utilised in prediction models. The prediction model can take this floor plan as an input and compose a directed graph of the possible routes of the user. If the current position of the user is known, a speculative adaptation task can use this prediction model and try to open connections to APs that are available in the next possible locations first. This all can shorten the disconnection time when the user is moving inside the building.

We implemented alternative tasks to open connections to BT access points. Firstly, we implemented a task that scans for access points, opens a connection to an AP that provides the needed service, and finally delivers the connection in its response (Figure 75).

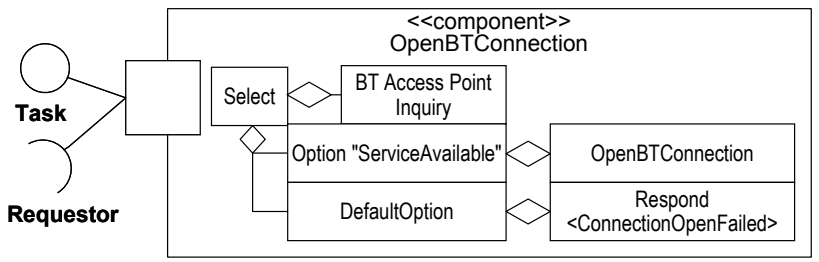


Figure 75. A task that scans for access points and opens a connection to a found access point.

Secondly, we implemented a speculative adaptation task that tries to open connections to BT access points that are available in a specific environment (Figure 76).

A composition schema defines the task and the addresses of known APs. It is fetched (e.g. from a BT access point) at runtime, a task factory is initialised for it, and finally a task is requested to open BT connections to the next known access points when the user is moving in a specific physical space. The task arranges the address list so that the addresses of the most used APs are at the beginning of the list. Then, it tries to open connections to the addresses in the list. It does not need to scan for APs (that requires typically more than ten seconds) but it tries to open connections directly to the known APs. If a

connection is successfully opened, the prediction model is updated and the connection is delivered in the response of the task.

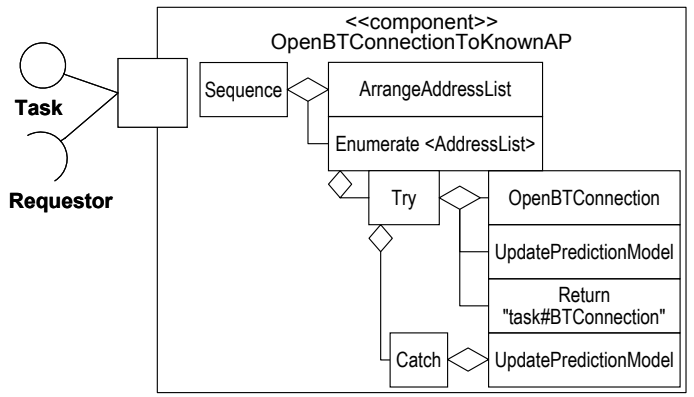


Figure 76. A task that tries to open connections to the known BT access points.

Thirdly, we implemented actions that were used in composition schemas. We implemented a *BTConnectionOpen* action that can call the *BluetoothConnector* framework to scan for BT access points and to open connections to known BT access points. Furthermore, we implemented the *AddressListArranger* action to arrange BT address lists so that the addresses of the most frequently used access points are at the beginning of the list.

7.5.2 Experiment setup

We made an experiment setup in order to evaluate how speculative adaptation tasks can improve the usage of BT access points in a real usage environment. We used two Java MIDP enabled Nokia N70 mobile phones, which worked as BT access points. Both mobile phones had running servers that provided contents for the requestors. The presented execution times were measured from the system clock. However, it must be noted that the connection opening time and data transmission speed can fluctuate in a Bluetooth network and thus affect the measurement results. In addition, it must be noted that BT connection opening can fail although the access point is in the range of the BT device. Furthermore, the BT device may not always find all the APs that are in the range of the BT device. These issues are not considered in the presented calculations.

The background processes that are running in the mobile device may also affect execution times of the processing threads, too. We repeated the measurements three times in order to minimise the measurement errors. Each presented measurement result is the average of the three separate measurement values. A thread pool that had 100 Java threads was used in the measurements.

7.5.3 Performance benefits

The average time to scan for services available via BT access points was 18.1 seconds in the Nokia N70 mobile phone. The opening of a BT connection took 0.8 seconds. Thus the total time to open a connection to a new BT access point was 18.9 seconds. In many cases this delay is far too long for spontaneous mobile usage. For example, this delay has a great effect on the usability of local services (see Section 7.4).

Speculative adaptation can shorten the connection open delay. The composition schema can define addresses for the access points and a task that tries to open BT connections to known addresses without making a service inquiry. If the access point is not in the range, connection cannot be opened.

The following usage scenario illustrates the benefits of speculative adaptation. A building may offer BT access points for the visitors. For simplicity, it can be defined that a single location in the building offers exactly one AP for the user.

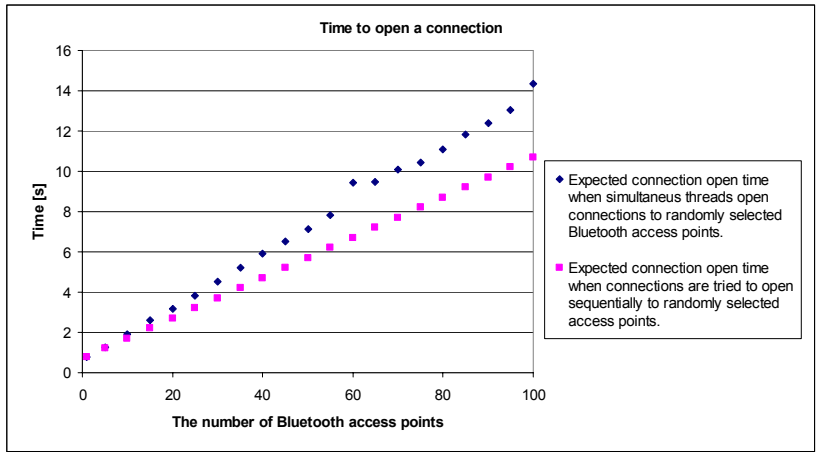


Figure 77. Time to execute multiple connection open threads.

Different strategies can be used when connections are opened to APs. We implemented first a test case in which multiple simultaneous threads were started to open connections to APs so that only one thread succeeded in opening a connection. As presented in Figure 77, the time to execute these threads increases linearly when the number of APs increases. In addition, the memory consumption rises as the number of threads increases.

Alternative solution is to try to open connections sequentially to randomly selected BT access points. The probability that a connection open will the first time is:

$$P_{fail} = \frac{n-1}{n}$$

where,

n = The number of BT access points.

As a result, the expected value of connection open time is:

$$t(n) = t_{copen} + \frac{n-1}{n} * t_{cfailure} + \frac{n-1}{n} * \frac{n-2}{n-1} * t_{cfailure} + \frac{n-1}{n} * \frac{n-2}{n-1} * \frac{n-3}{n-2} * t_{cfailure} \cdots$$

$$= t_{copen} + \sum_{i=1}^{n-1} \frac{(n-1)!*(n-i)!}{(n-1-i)!*n!} * t_{cfailure}$$

$$= t_{copen} + \sum_{i=1}^{n-1} \frac{(n-i)!}{(n-1-i)!*n} * t_{cfailure}$$

$$= t_{copen} + \sum_{i=1}^{n-1} \frac{n-i}{n} * t_{cfailure}$$

$$= t_{copen} + \frac{n-1}{2} * t_{cfailure}$$

where,

t_{copen} = Connection open time to an access point that is in the range of the mobile device.

$t_{cfailure}$ = Time to notice that the connection open to an access point that is not in the range of the mobile device failed.

The average time to notice that a single connection open failed was 0.2 s in the Nokia N70 device. Figure 77 shows the calculated expected values of

connection open times for these measured values. As can been seen, the expected value of a connection open time is shorter when BT connections are opened sequentially. However, if the number of access points is more than 182, it is faster to use the inquiry mechanism.

A mechanism that is capable of predicting the next access point that will be in the range of the mobile device can make speculative adaptation faster for a greater number of access points. For example, the user may use certain routes inside the building. The application can learn the routes of the user and predict the next BT access points, which will be in the range of the mobile device. If a prediction mechanism is available, the expected value for the connection open time can be calculated with the following formula:

$$t(n) = t_{copen} + \sum_{i=1}^{n-1} (1 - \mu)^i * t_{cfailure}$$

where μ is the prediction accuracy ($0 \le \mu \le 1.0$). Figure 78 shows how prediction accuracy affects the expected value of the connection open time. With 100 access points the expected value of the connection open time is 1.4 seconds when the prediction accuracy is 10 percent. When the prediction accuracy is 50 percent the expected value of the connection open time is reduced to 1.0 seconds.

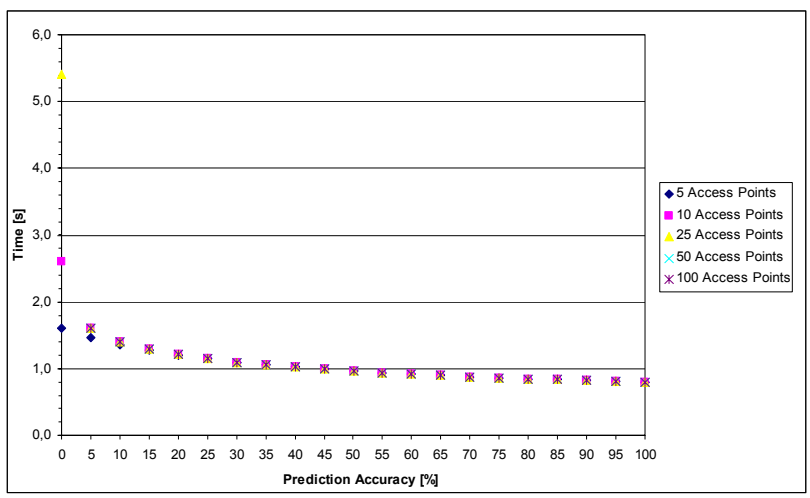


Figure 78. The expected value of the connection open time when the next access point can be predicted (t_{copen}=0.8 s, $t_{cfailure}$=0.2 s).

7.5.4 Implementation benefits

The speculative task-based BT connection opening requires changes both on the client and server-side. The server-side must provide the addresses of BT access points that are available in a certain area. A task that tries to open connections to known access points is needed, too. Finally, an implementation for the task-based composition technique and actions that can perform the composition assignments defined in the composition schemas must be installed to the client (Figure 79).

Action	Description	Size [lines of code]
AddressListArranger	Composes an address list for Bluetooth access points.	50
BTConnectionOpen	Scans for Bluetooth access points and opens connections to the found access points.	120
	Total	170 lines

Reference Implementation	Description	Size [lines of code]
XMLImport	Opens a connection to defined URI, parses the fetched XML document, and finally composes a data object presentation for it.	145
TaskFactoryImpl	An action that is able to initialise a task factory.	35
Select, Sequence, Enumerate, and Try-Catch action structures	These action structures were used in the composition schemas.	576
	Total	756 lines

Figure 79. The action implementations used in the speculative adaptation task.

The *sequence*, *request*, *enumerate*, *refresh*, and *if-else* execution structures of TaskCAD were used in the connection open tasks. Thus a total 756 lines of code of TaskCAD were used in the example. As can be seen, the task-based composition technique reduced the total coding effort by 82 percent in this example.

7.5.5 Summary

As presented in this case study, a mechanism that offers good prediction accuracy (see Section 2.2.4) can shorten the expected value of the BT connection open time and improve the usability of local services. However, it must be noted that it takes time to download a composition schema for a speculative BT connection open task. For example, downloading a composition schema that defined BT addresses for 10 access points (size was 3.1 Kbytes) and initialising a task factory with it took total 1.5 seconds in the Nokia N70 device.

8. Comparisons to related work

Chapter 3 gives an overview of methods supporting dynamic composition of component-based adaptive applications. This chapter compares the task-based composition technique to other methods supporting dynamic composition of component-based adaptive applications. It is organized as follows. The introduction is in Section 8.1. Examples of architectures and frameworks that support adaptation of component-based applications are discussed in Section 8.2. Techniques that provide languages for dynamic component-based composition and adaptation are discussed in Section 8.3. Finally, the content and context-sensitive applications are discussed briefly in Section 8.4.

8.1 Introduction

The goal of the task-based composition technique is to make the dynamic composition of content and context-sensitive applications more fluent and to help developers to implement adaptive applications for mobile devices. It provides many features that promote implementation of applications that can fluently adapt for new contexts. Firstly, the used adaptation policies (composition schemas) and actions can be replaced with new ones at runtime. Secondly, it facilitates the caching of both adaptation tasks and application instances. Thirdly, it supports speculative adaptation, context-sensitive handling of errors, utilisation of new execution structures in adaptation, and provides structures that enable the user to control the adaptation.

A number of techniques have been developed for overcoming compositional mismatches between components. For example, a lot of work has been carried out related to formal models (e.g. [BPR02, Pah02, ITLS04, BBC05]), to wrapping techniques (e.g. [GHJV95, YeS97, BCK98]), and to intermediate forms (e.g. [Bea92, Pur94, OMG96, McA95, Rog97]). The task-based composition technique does not focus on solving computational mismatches between components but supports dynamic composition of content and context-sensitive applications of components that are *plug compatible* [ALSN01] (see Section 3.2) with other components.

The context-aware features can be classified into *contextual sensing*, *adaptation*, *resource discovery*, and *augmentation* features [Pas98] (see Section 2.1). The task-based composition technique focuses on adaptation features. In addition, it supports caching of content and context-sensitive application instances and can in this manner facilitate the implementation of resource discovery features.

Adaptation can also be divided into static or dynamic adaptation [SaM03] (see Section 2.2.1). The task-based composition technique supports dynamic adaptation and fine-tuning of adaptive application. In addition, dynamic adaptation techniques can be divided into *parametric* and *compositional* adaptation techniques [MSKC04] (see Section 2.3). The task-based composition technique supports both parametric and compositional adaptation of component-based applications.

Task modelling and analysis is made in many sources [JJWS98, WaG00, GPSS04]. However, this research typically focuses on the tasks of the user and investigates what people do when they carry out one or more tasks and involves collecting information about how people perform those tasks [JJWS98]. This dissertation focuses on tasks, which are performed by computing devices. Thus this work is not in the scope of this dissertation and is not discussed in more detail.

8.2 Architectures, frameworks, and structures for component-based adaptive applications

Many architectures and frameworks (e.g. [DPH91, FuT99, AmW99, BCS02, Gri04]) are developed to provide solutions for component-based adaptive applications (see Chapter 3). Unlike these solutions, the task-based composition technique does not concentrate on the structure of the adaptive application but aims to make the dynamic composition of adaptive content and context-sensitive applications more fluent.

Middleware solutions for reactive (publish-subscribe, Odyssey, and Gaia) and proactive adaptation (e.g. Carisma [Cap03]) exist. However, these solutions do not offer support for speculative adaptation, where content and context-sensitive application parts are composed in multiple phases for potential future contexts.

The task-based composition technique enables developers to extend composition schemas with new tasks and actions that can predict potential future contexts and request tasks to perform speculative adaptation actions and to compose application parts for the predicted contexts. The composed content and context-sensitive parts can be cached and possibly utilised in the future.

Middleware solutions typically offer a set of services that support construction of certain kinds of applications. The middleware approach can support application transparent adaptation [BFK+00] but is suitable only for programs that are written against a specific middleware platform [MSKC04b]. The focus of this dissertation is on techniques adapting the application itself. Thus the majority of the middleware solutions are not directly in the scope of this dissertation and are not discussed in greater detail.

In a traditional Object-Oriented Programming (OOP) developers implement objects that typically define data structures of data types and operations that can be applied to data structures [BWL03]. As is the case in Facet-based programming (FBP) [BWL03], in task-based composition functionality and data are clearly separated. Adaptation actions do not contain any data but they fetch application instances from the defined data sources and compose application instances to the defined targets.

Several solutions (e.g. Linda [CaG89], LIME [MPR01], TOTA [MaZ04], and One.world [Gri04]) offer a shared memory space that exploits localized data structures in order to let processing components gather information, interact, and coordinate with each other [MaZ04]. However, these solutions do not support sharing of context-sensitive data. In other words, they do not provide methods that enable processing components to fetch the most suitable data for the context. The task-based composition technique promotes the sharing of context-sensitive instances with *context comparators* that are capable of calculating how suitable an instance is for a processing context. Various kinds of context comparators can be defined for the instances saved to the shared memory (environments). As a result, the most suitable data can be selected for the PC and delivered for processing components.

8.3 High-level programming techniques for context-sensitive component-based composition and adaptation

This section concentrates on the most important techniques, which enable high-level programming for adaptation strategies and application components and separate reconfiguration schemas from the application. These techniques are discussed in more detail in Chapter 3.

In the same manner as these techniques, the task-based composition technique also separates configuration concerns from the business logic of an application. However, unlike the task-based composition technique, these discussed solutions do not offer direct support for speculative adaptation. The task-based composition technique enables an application to be composed in many phases and supports caching of both content and context-sensitive instances and adaptation tasks. Asynchronous speculative adaptation tasks can compose application instances in the background without blocking the application. Furthermore, unlike the described techniques, TaskCAD offers execution structures for adaptation actions that enable developers to define context-sensitive handling for errors raised while an application is composed, inform the user about the progress of the adaptation, and enable the end-user to control the adaptation. In addition, developers can extend the task-based composition technique with implementations that can provide new execution structures for adaptation actions.

8.4 Client-side solutions for adaptive content and context-sensitive applications

Much research has been carried out in the domain of adaptive content and context-sensitive applications (see Section 3.6). The task-based composition technique supports adaptation, in which it is possible to combine information coming from separate sources in a single context-sensitive UI on the client-side. Unlike the task-based composition technique, the described approaches do not offer direct support for speculative adaptation in which the previously composed content and context-sensitive application instances can be utilised when an application is composed for new contexts.

The dynamic document [KPT94] and Ajax [Gar05] approaches support information pre-fetching and background processing. However, unlike the task-based composition technique, these solutions do not focus on component-based composition but are rather designed for client-side content adaptation. ICrafter is a server-side approach that does not support speculative adaptation or utilisation of previously prepared application instances in adaptation. The *universal interactor* solution uses information pre-fetching to speed up UI composition [HKSR97]. The task-based composition technique does not introduce any fixed speculative adaptation method (like information prefetching) but facilitates utilisation of different kinds of adaptation actions and prediction models in speculative adaptation and enables the adaptation policies and actions to be changed, downloaded, and configured with context-sensitive parameters at runtime. For example, a composition schema that defines a speculative adaptation task for a building can be downloaded at runtime to minimise the connection open times when local Bluetooth service access points are used, to do information prefetching, and to compose UIs for local services in the background. In addition, the composition schema can have structures that enable the end-user to control adaptation and structures that define context-sensitive handling for errors raised while the UIs are composed.

8.5 Summary of main contributions with respect to existing solutions

The task-based composition technique supports both parametric and compositional adaptation of component-based applications. Unlike the middleware approaches, the task-based composition technique focuses on adapting the application itself and supports dynamic composition of content and context-sensitive applications of components that are *plug compatible* [ALSN01] with other components. In addition, it does not concentrate on the structure of the adaptive application but aims to make the dynamic composition of adaptive content and context-sensitive applications more fluent.

Several solutions (e.g. Linda [CaG89], LIME [MPR01], TOTA [MaZ04], and One.world [Gri04]) offer a shared memory space that exploits localized data structures in order to let processing components gather information, interact, and coordinate with each other [MaZ04]. However, these solutions do not support

sharing of context-sensitive data and do not provide methods that enable processing components to fetch the most suitable data for the context. The task-based composition technique promotes the sharing of context-sensitive instances with *context comparators* that are capable of calculating how suitable an instance is for a processing context. Various kinds of context comparators can be defined for the instances saved to the shared memory (environments). As a result, the most suitable data is selected for the PC and delivered for the processing components.

Several techniques provide languages for adaptation and thus separate configuration concerns from the business logic of an application. However, unlike the task-based composition technique, the discussed solutions do not offer direct support for speculative adaptation. The task-based composition technique does not introduce any fixed speculative adaptation method (like information prefetching) but facilitates utilisation of different kinds of adaptation actions and prediction models in speculative adaptation and enables the adaptation policies and actions to be changed, downloaded, and configured at runtime.

The task-based composition technique enables an application to be composed in many phases and supports caching of both content and context-sensitive instances and adaptation tasks. As a result, the previously composed content and context-sensitive application instances can be utilised when an application is composed for new contexts. The asynchronous speculative adaptation tasks can compose application instances in the background without blocking the application. Furthermore, unlike the discussed language-based techniques, TaskCAD offers execution structures for adaptation actions that enable developers to define context-sensitive handling for errors raised while an application is composed, inform the user about the progress of the adaptation, and enable the end-user to control the adaptation. In addition, developers can extend the task-based composition technique with new implementations providing new execution structures for adaptation actions.

9. Conclusion

This dissertation describes the task-based composition technique and presents how it can be utilised in the domain of adaptive content and context-sensitive applications. This chapter is organized as follows. Research problems and how the set requirements (see Section 1.2.5) are materialized in the implementation of the task-based composition technique are summarized first in Section 9.1. Then, the contributions of this dissertation are reviewed in Section 9.2. Future research directions are discussed in Section 9.3. Finally, concluding remarks are given in Section 9.4.

9.1 Task-based composition technique as a platform of adaptive applications

The concrete implementation of the task-based composition technique, called TaskCAD, is discussed in Chapter 6. It is a universal and extendable solution supporting both reactive and proactive adaptation and separating composition concerns clearly from the business logic of an adaptive application. The XML-based composition schemas can define tasks and context-sensitive actions and their parameters and contents. This all makes it easier for the developers to implement adaptive applications for mobile devices. In the best case, if ready-made adaptation actions are available, implementation of an adaptive application will require no coding effort at all but only the creation of new composition schemas. In addition, it may be possible to reuse parts of the existing composition schemas in new adaptive applications. Composition schemas can be downloaded and modified at runtime without changing the code performing the actual composition actions. As a result, it is possible to change the adaptation strategy at runtime.

The task-based composition technique offers methods to make the dynamic composition of content and context-sensitive applications more fluent. It offers methods that support:

1. **Dynamic adaptation strategies**. The adaptation policies (composition schemas) and actions can be changed, downloaded, and configured at runtime.
2. **Speculative adaptation.** The task-based composition technique does not introduce any fixed speculative adaptation method (like information prefetching) but facilitates utilisation of different kinds of adaptation actions and prediction models in speculative adaptation.
3. **Caching of adaptation tasks.** Many parts of the application can request the same adaptation tasks. Thus the caching of tasks can speed up the adaptation process by increasing utilisation of previously started tasks. However, it is not trivial to manage the cache of context-sensitive tasks which can have dependencies on other tasks, too. The challenge is to identify when a task is suitable for a specific request and Processing Context (PC). A part of the cached tasks may have expired. In addition, in order to minimize the usage of the limited memory of a mobile device, those tasks deemed not suitable for the current or possible forthcoming PCs must be recognized, stopped, and finally removed from the cache. The task-based composition technique offers methods that facilitate caching of context-sensitive tasks and are able to select the most suitable tasks for a request and PC, and finally enable the application to utilise them in adaptation. Furthermore, it offers methods that are able to recognize and stop those started adaptation tasks that are not suitable for the current or predicted contexts.
4. **Caching of context-sensitive application instances.** The tasks can compose applications in multiple phases and utilise cached application instances. For example, in order to improve usability, alternative context-sensitive presentations can be offered for a single Web page. A client-side task can fetch a document source that contains content elements for various contexts from the Web, parse it, and finally store the parsed document in the cache. As a result, client-side tasks can compose new UIs for the context-sensitive elements of the cached documents and display them for the end-user. This all can be made without causing network traffic and without parsing the document sources again. As a result, adaptation is faster and does not cause costs for the end-user.
5. **User-directed adaptation**. The reference Java implementation of the task-based composition technique offers ready-made executions for adaptation

actions that enable the end-user to control adaptation. For example, the end-user can accept or select between alternative adaptation actions. In addition, the composition schemas can define context-sensitive feedback messages and actions that will display them and notify the end-user about progress of adaptation.

6. **Context-sensitive handling of errors.** A composition schema can define context-sensitive handling for errors raised while the adaptation actions are performed.
7. **Utilisation of new execution structures in adaptation.** Developers can implement new execution structures for adaptation actions and use them in composition schemas. For example, new conditional executions that utilise data available in the application environment and control execution of the defined adaptation actions can be implemented and utilised in task-based adaptation.

The research problems were the following:

- *How to support active and passive context-awareness in dynamic composition of adaptive applications?*
- *How to support context-sensitive handling of errors appearing in the dynamic composition of adaptive applications?*
- *How to support speculative adaptation in dynamic composition?*
- *How to support dynamic client-side composition of adaptive mobile browsers?*

The general quality goals of the technique were discussed in Section 1.2.5. Based on the experiences gained from different use cases and examples, the following compares the task-based composition technique to these goals:

- **Generic**. The core of the task-based composition technique is based on the content adaptation model introduced by W3C. As shown in Section 4, the core of the technique describes a generic structure that supports the dynamic component-based composition of adaptive applications with context-sensitive adaptation tasks and actions. The core of the generic model can be extended to support dynamic composition of

various kinds of context-sensitive component-based applications. In addition, it can be implemented with various kinds of object-oriented programming languages, for example, with Java and C++.

- **Extensible**. As shown in Section 6, TaskCAD does not only provide a predetermined and fixed set of methods supporting dynamic composition; instead it allows developers to extend the technique with new kind of actions, settings, and context description elements supporting dynamic composition of different kinds of adaptive applications. For example, as discussed in Section 4.6, developers can extend the technique with implementations introducing new execution structures for adaptation actions. Since the technique is highly extendable, it can be specialised to support dynamic composition of a wide range of adaptive applications.
- **Policy independence**. The task-based composition technique concentrates on the composition concerns of component-based adaptive applications. The core of the technique does not prescribe policies, constraints, services, or facilities that are specific to particular application domains, or deployment environments. Instead the core of the technique describes generic mechanisms that support the dynamic composition of various kinds of adaptive content and context-sensitive applications of software components.
- **Scalable.** The task-based composition technique is designed to support adaptation in very different kinds of environments and devices offering processing and memory capabilities of a different level. The interfaces and classes of the Java reference implementation implement the core parts of the technique only. As a result, the core implementations have a small memory footprint, which makes the technique applicable also in mobile devices with a limited memory. TaskCAD offers the interfaces and classes for the task-based composition technique. Its compiled size is currently 120 Kbytes and it has been used in the Java 2 Standard Edition and Java MIDP environments and in devices having hundreds of Kbytes of memory for the Java technology stack.
- **Separation of concerns**. As discussed in Chapter 6, TaskCAD offers an XML-based language for composition schemas. As a result, the adaptation concerns are clearly separated from the business logic of the

application. The composition schemas can be changed at runtime and possibly reused in various applications. In addition, the XML-based composition schemas can be downloaded from the Web and executed on the new environments that support the execution of tasks.

- **Incremental**. The task-based composition technique enables developers to utilise various kinds of adaptation components in the composition schemas. In addition, composition schemas can utilise the tasks of other composition schemas and so it is possible to make incremental development for composition concerns of adaptive applications. As discussed in Section 4.6, the actions of a composition schema can dynamically download composition schemas, initialise task factories for them, and finally request the initialised task factories to execute the adaptation tasks.
- **High performance**. The task-based composition technique does not cause considerable overhead for dynamic composition of component-based adaptive applications. The results of the measurements (see Section 5.1) indicate that using of the task-based composition technique does not significantly disturb the end-user of an adaptive application. In addition, because the technique makes it easier for developers to implement component-based adaptive applications, they can concentrate more on improving the performance and quality of the components of adaptive applications.

MIMEFrame defines architecture for adaptive user agents and browsers (see Section 7.2). The reference Java implementations of MIMEFrame enable developers to compose separate user agents or browsers embedded in other mobile applications of consistent and reusable components. As a result, instead of implementing the whole browser from scratch, developers can put more effort into improving the browser components. The task-based composition technique enables browsers to be dynamically composed for different contexts.

The experiences gained from the discussed case studies suggest that the task-based composition technique facilitates the implementation of content and context-sensitive applications. Firstly, it decreases the coding effort. Secondly, it supports the reuse of composition schemas and enables adaptation policies to be modified or replaced at runtime to improve efficiency and usability. For

example, as shown in Section 7.5, tasks can be used to minimize the disconnection time when the services of Bluetooth access points are utilised. It is easy to extend existing composition schemas with new adaptation actions and tasks. In addition, the use cases proved that it facilitates dynamic composition of applications that utilise data coming from various sources. Thirdly, the technique supports caching of tasks and content and context-sensitive application instances and speculative adaptation where application is composed in multiple phases for rapidly changing contexts.

However, it must be noted that the task-based composition technique does set requirements for the target mobile device. Firstly, the reference Java implementation of the task-based composition technique can be used only in devices that support thread-based processing and provide hundreds of kBytes of memory for the Java technology stack. Secondly, implementations that are able to download and parse the XML-based composition schemas and to execute the tasks of downloaded composition schemas must be installed to the client device. Thirdly, plugin implementations for the used actions must be installed to the client.

The adaptation tasks can do concurrent processing. This sets requirements for adaptation action implementations. For example, synchronization problems may arise if multiple tasks modify the same data. In order to avoid synchronization problems, an adaptation action must lock the application instances that it modifies. Furthermore, the execution order of the code can change in concurrent task-based processing. As a result, it may be more difficult to test the code performing the adaptation actions. Although the task-based composition technique does not directly solve the difficulties related to concurrent processing, it facilitates modification of composition schemas. As a result, it is faster to do edit and test cycles for target applications. For example, a test task can be defined to log the raised errors. It can have an action that will request defined adaptation tasks randomly and a try-catch action structure that will log the error responses of the requested tasks.

The reuse of adaptation actions requires well-documented action plugins. The functionality of adaptation action components must be clearly described. In addition, it must be defined what kind of application instances an action needs and what kind of instances it will produce to the defined targets. The task-based composition technique does not directly support testing of various kinds of

component configurations. However, it helps developers to implement test beds that can test reactive, proactive, and speculative adaptation of target applications.

In contrast to a standard browser and adapted content approach, the implementation of content and context-sensitive mobile browsers requires more effort. However, usability can be improved if the browser is optimised to offer controls and views that are optimised for the specific contents and for services available on the Web. Furthermore, it is possible to embed the browser to an adaptive mobile application. The server-side can provide a composition schema for a task that can compose optimised UIs for the provided services. The composition schema can be downloaded to the client that can execute its tasks. The tasks can combine a single UI for context-sensitive content elements. If the content elements are available in the cache, tasks can compose and adapt UIs on the client-side without causing network traffic and costs for the mobile user.

However, although MIMEFrame and the task-based composition technique facilitate implementation of adaptive browsers, it may be too challenging for developers lacking the necessary programming experience to construct totally new adaptive browsers. However, if a lot of ready-made browser components and composition schemas are available, it is possible to implement new kinds of browsers without manual coding. Only composition schemas have to be edited. This can be made with standard text or XML editors. In addition, the mPlaton editor facilitates implementation of the composition schemas.

9.2 Summary of contributions

Contributions of this dissertation were enumerated in Section 1.4. The main contributions are the following:

- *The development of the task-based composition technique concept that supports reactive, proactive, and speculative adaptation of content and context-sensitive applications.* As discussed in Chapter 4, and shown with case studies, the task-based composition technique supports dynamic composition of adaptive content and context-sensitive applications. It offers structures that enable developers to define context-sensitive handlings for errors raised during an application is adapted for

a new context and enables end-users to control execution of adaptation actions. In addition, TaskCAD enables developers to define context-sensitive feedback messages that can inform the end-user of the progress of adaptation. As discussed in Section 4.5, the task-based composition technique supports both caching of asynchronous adaptation tasks and context-sensitive application instances composed with tasks. As a result, the task-based composition technique can be used in speculative adaptation by requesting tasks to compose application instances for the predicted processing contexts.

- *The development of the MIMEFrame framework that defines architecture for adaptive user agents and browsers.* The MIMEFrame framework offers interfaces and components for adaptive user agents and browsers (see Section 7.2) that can be composed with tasks. The author has made Java implementations for the MIMEFrame framework (Chapter 7).

The task-based composition technique is evaluated with case studies. Related to this, the author has carried out:

- *The Java reference implementation for the task-based composition technique*, called TaskCAD, offers core interfaces and classes for executing adaptation tasks with threads. Furthermore, it offers an interpretable XML-based language for composition schemas. As a result, both the components and adaptation policies can be changed dynamically so that they could better correspond to possible emerging new requirements. At the same time, the adaptation concerns are clearly separated from the business logic of the application. TaskCAD offers various kinds of executions for adaptation actions. For example, it offers sequential and exclusive executions and request-adapt, try-catch, accept, select, enumerate, and refresh structures for actions. In addition, it enables developers to use specialized action composites in composition schemas.
- *A tool that facilitates implementation of XML-based composition schemas*. The implemented mPlaton editor (see Section 6.5) facilitates implementation of XML-based composition schemas.

- *The analysis of case studies that were implemented for mobile devices.* As discussed in Chapter 4, and shown with case studies, the task-based composition technique can be utilised in the domain of adaptive browsers. Three case studies were implemented for mobile devices and evaluated. The discussed case studies shown that the task-based composition technique decreases the implementation effort and speeds up adaptation (see Chapter 7).

9.3 Future work

The task-based composition technique provides a platform for experimenting with adaptation of component-based applications. TaskCAD is a prototype for the task-based composition platform and language. Here listed are some directions for the future work:

More analysis and evaluation for the task-based composition technique. The presented case studies show that the task-based composition technique facilitates implementation of content and context-sensitive applications and evaluate both performance and implementation benefits of the task-based composition technique. However, more analysis and evaluation is still needed for the task-based composition technique. It is important to notice that the given evaluations do not compare TaskCAD to other available adaptation methods but only show what kind of implementation and performance benefits it provided in the adaptive browser implementations. Thus future research and experiments are still needed to provide more numerical measurement information about the task-based composition technique and alternative adaptation methods. For example, more information about the performance and implementation benefits of various adaptation methods is needed. In addition, it is important to notice that there are many parameters that affect the performance of the task-based composition. For example, the size of the thread pool and the size of the task and application instance caches have an effect on the performance. New measurements are needed to evaluate the effect of these parameters in various execution environments.

New components and interfaces that can be used in various kinds of component configurations. Dynamic component-based composition requires high quality

components that can be used in various kinds of configurations and replaced on the fly. Clear and uniform interfaces are required for component implementations. In addition, techniques that help developers to ensure that various component configurations will work correctly in different contexts are needed.

Different tools supporting the construction of composition schemas. In order to make utilisation of the task-based composition technique easier, new tools can be implemented to enable also people without programming skills to modify composition schemas. For example, tools like wizards can enable the end-users to modify the composition schemas at runtime in a mobile device.

Different integrations. The core of the task-based composition technique is implemented with Java. However, the technique can be implemented with other object-oriented languages, like C++, too. New implementation can be provided to enable developers to use the task-based composition technique in new application environments.

New context prediction models and requestors for adaptation. The speculative adaptation requires mechanisms predicting possible forthcoming processing contexts. New prediction components that utilise the context history information and improve prediction accuracy can improve the efficiency of speculative adaptation. For example, fuzzy logic could be used in processing multidimensional context history information.

Generative methods for integrating the adaptation schemas in the application code. TaskCAD is applicable in Java enabled mobile devices that have hundreds of Kbytes of memory for the Java technology stack. Generators that are capable of integrating context-sensitive component composition schemas directly to the code of an adaptive application can decrease memory requirements. Less classes and interfaces are needed, which makes the installation package of an adaptive application smaller. As a result, the task-based composition technique could be used in devices having less memory for the Java technology stack. However, it must be noted that if the adaptation strategies are integrated directly into application components, it is difficult to change the adaptation strategies at runtime.

New case studies. The task-based composition technique has been mainly used in the domain of adaptive content and context-sensitive applications. New case studies with new kinds of adaptation actions are needed to test the applicability of the technique in different application domains.

9.4 Concluding remarks

The task-based composition technique helps software developers to implement adaptive content and context-sensitive applications for mobile devices. The composition policies can be defined at a higher-level with the task-based composition language. As a result, the composition concerns are clearly separated from the application business logic. In addition, because it is an interpretable language, it is possible to change composition schemas and adaptation strategies at runtime. Furthermore, composition schemas can be reused in new applications. To enable task-based composition, the presented system provides a tool that supports construction of composition schemas and an environment that is capable of running them.

References

[AAH+97] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton, "Cyberguide: A mobile context-aware tour guide," *Wireless Networks*, vol. 3, pp. 421–433, 1997.

[AAH05] P. Alahuhta, J. Ahola, and H. Hakala, "Mobilizing Business Applications – A survey about the opportunities and challenges of mobile business applications and services in Finland," *TEKES Technology Review 167/2005*, 2005.

[ABB+89] G. Attardi, C. Bonini, M. R. Boscotrecase, T. Flagella, and M. Gaspari, "Metalevel Programming in CLOS," *Proceedings of European Conference on Object-Oriented Programming (ECOOP'89)*, 1989.

[ABFG04] D. Austin, A. Barbir, C. Ferris, and S. Garg, "Web Services Architecture Requirements," *W3C Working Group Note*, 2004.

[ACM04] A. Arsanjani, F. Curbera, and N. Mukhi, "Manners externalize semantics for on-demand composition of context-aware services," *Proceedings of IEEE International Conference on Web Services*, pp. 583–590, San Diego, California, USA, 2004.

[AcN01] F. Achermann and O. Nierstrasz, *Applications = Components + Scripts – A tour of Piccola*: Kluwer, 2001.

[ALSN01] F. Achermann, M. Lumpe, J.-G. Schneider, and O. Nierstrasz, "PICCOLA – a Small Composition Language," *Formal Methods for Distributed Processing: A Survey of Object-Oriented Approaches*, H. Bowman and J. Derrick, Eds.: Cambridge University Press, pp. 403–426, New York, USA, 2001.

[AmW99] N. Amano and T. Watanabe, "LEAD++: An Object-Oriented Reflective Language for Dynamically Adaptable Software Model," *IEICE TRANSACTIONS on Fundamentals of*

Electronics, Communications and Computer Sciences, vol. 82, pp. 1009–1016, 1999.

[AsS03] D. Ashbrook and T. Starner, "Using GPS to learn significant locations and predict movement across multiple users," *Personal and Ubiquitous Computing*, vol. 7, pp. 275–286, 2003.

[ATB04] D. Ayed, C. Taconet, and G. Bernard, "Deployment and Reconfiguration of Component-based Applications in AMPROS," *presented at Proactive computing workshop (PROW 2004)*, Finland, 2004.

[AzJ00] B. Aziz and C. Jensen, "Adaptability in CORBA: The Mobile Proxy Approach," *Proceedings of the 2nd International Symposium on Distributed Objects and Applications (DOA '00)*, pp. 295–304, Antwerp, Belgium, 2000.

[BBC05] A. Bracciali, A. Brogi, and C. Canal, "A formal approach to component adaptation," *The Journal of Systems & Software*, vol. 74, pp. 45–54, 2005.

[BCA+00] G. S. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran, N. Parlavantzas, and K. Saikoski, "A Principled Approach to Supporting Adaptation in Distributed Mobile Environments," *Proceedings of International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE 2000)*, pp. 3–12, Limerick, Ireland, 2000.

[BCFH03] N. O. Bouvin, B. G. Christensen, K. G. Frank, and A. Hansen, "HyCon: a framework for context-aware mobile hypermedia," *New Review of Hypermedia and Multimedia*, vol. 9, pp. 59–88, 2003.

[BCK98] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*: Addison–Wesley, 1998.

[BCS02] E. Bruneton, T. Coupaye, and J. B. Stefani, "Recursive and Dynamic Software Composition with Sharing," *Proceedings of Seventh International Workshop on Component-Oriented Programming (WCOP)*, Malaga, Spain, 2002.

[BCS03] D. Bonino, F. Corno, and G. Squillero, “Dynamic prediction of Web requests,” *Proceedings of the IEEE Congress on Evolutionary Computation*, Canberra, Australia, 2003.

[Bea92] B. W. Beach, "Connecting Software Components with Declarative Glue," *Proceedings of the International Conference on Software Engineering (ICSE ’92)*, pp. 120–137, Melbourne, Australia, 1992.

[BeA01] L. Bergmans and M. Aksits, "Composing crosscutting concerns using composition filters," *Communications of the ACM*, vol. 44, pp. 51–57, 2001.

[BEF+00] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, X. Ya, and Y. Haobo, "Advances in network simulation," *Computer*, vol. 33, pp. 59–67, 2000.

[BEK+00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," *W3C Note*, 2000.

[Ben86] J. L. Bentley, "Programming pearls: Little languages," *Communications of the ACM*, vol. 29, pp. 711–721, 1986.

[BFK+00] B. Badrinath, A. Fox, L. Kleinrock, G. Popek, P. Reiher, and M. Satyanarayanan, "A Conceptual Framework for Network and Client Adaptation," *Mobile Networks and Applications*, vol. 5, pp. 221–231, 2000.

[BGGW02] M. Butler, F. Giannetti, R. Gimson, and T. Wiley, "Device Independence and the Web," *IEEE Internet Computing*, vol. 6, pp. 81–86, 2002.

[BHL01] I. Ben-Shaul, O. Holder, and B. Lavva, "Dynamic Adaptation and Deployment of Distributed Components in Hadas," *IEEE Transactions on Software Engineering*, vol. 27, 2001.

[BJK03] A. Brown, S. Johnston, and K. Kelly, "Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications," *Rational-IBM White paper*, 2003.

[BlC97] G. S. Blair and G. Coulson, "The case for reflective middleware," *Proceedings of 3rd Cabernet Plenary Workshop*, Rennes, France, 1997.

[BLLJ98] B. Bos, H. W. Lie, C. Lilley, and I. Jacobs, "Cascading Style Sheets, level 2 CSS2 Specification," *W3C Recommendation*, 80 p., 1998.

[BMR+96] F. Buschmann, R. Meunier, Hans Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*: Wiley, 1996.

[Bos99] J. Bosch, "Superimposition: A component adaptation technique," *Information and Software Technology*, vol. 41, 1999.

[BPR02] A. Brogi, E. Pimentel, and A. M. Roldán, "Compatibility of Linda-based Component Interfaces," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 66, pp. 1–15, 2002.

[BPS00] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0," *W3C Recommendation*, 2000.

[Bra92] G. Bracha, "The Programming Language Jigsaw: Mixins, Modularity and Multiple Inheritance," in *Department of Computer Science*: University of Utah, 1992.

[Bro95] K. Brockschmidt, *Inside OLE*: Microsoft Press Redmond, WA, USA, 1995.

[BWL03] N. M. Belaramani, C.-L. Wang, and F. C. M. Lau, "Dynamic Component Composition for Functionality Adaptation in Pervasive Environments," *Proceedings of the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS2003)*, San Juan, Puerto Rico, USA, 2003.

[CaG89] N. Carriero and D. Gelernter, "Linda in context," *Communications of the ACM*, vol. 32, pp. 444–458, 1989.

[CaL90] P. Calder and M. Linton, "Glyphs: Flyweight Objects for User Interfaces," *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 92–101, Snowbird, Utah, USA, 1990.

[Cap03] L. Capra, "Reflective Mobile Middleware for Context-Aware Applications," in *Department of Computer Science*: University of London, 2003.

[CBG+04] G. Coulson, G. S. Blair, P. Grace, A. Joolia, K. Lee, and J. Ueyama, "A Component Model for Building Systems Software," *Proceedings of IASTED Software Engineering and Applications (SEA'04)*, Cambridge, MA, USA, 2004.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," *W3C Note*, 2001.

[CDK00] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: concepts and design*. Boston, MA, USA: Addison–Wesley, 2000.

[CEM03] L. Capra, W. Emmerich, and C. Mascolo, "CARISMA: Context-Aware Reflective Middleware System for Mobile Applications," *IEEE Transactions on Software Engineering*, vol. 29, pp. 929–945, 2003.

[Cha91] E. Charniak, “Bayesian Networks without Tears,” *AI Magazine*, vol. 12, pp. 50–63, 1991.

[ChK00] G. Chen and D. Kotz, "A Survey of Context-Aware Mobile Computing Research," *Dartmouth Computer Science Technical Report TR2000-381*, 2000.

[CHRR04] L. Clement, A. Hately, C. von Riegen, and T. Rogers, "UDDI Version 3.0.2," *UDDI Specification Technical Committee Draft*, 2004.

[ChY97] K. Chinen and S. Yamaguchi, "An interactive prefetching proxy server for improvement of WWW latency," *Proceedings of the Seventh Annual Conference of the Internet Society (INET '97)*, Kuala Lumpur, Malaysia, 1997.

[CMD01] K. Cheverst, K. Mitchell, and N. Davies, "Investigating context-aware information push vs. information pull to tourists," *Proceedings of the Third International Workshop on Human Computer Interaction with Mobile Devices (Mobile HCI '01)*, Lille, France, 2001.

[CoK02] E. Cohen and H. Kaplan, "Prefetching the means for document transfer: a new approach for reducing Web latency," *Computer Networks*, vol. 39, pp. 437–455, 2002.

[Crn03] I. Crnkovic, "Component-based software engineering-new challenges in software development," *Proceedings of the 25th International Conference on Information Technology Interfaces (ITI 2003)*, pp. 9–18, Cavtat, Croatia, 2003.

[CSKO02] A. Corsaro, D. C. Schmidt, R. Klefstad, and C. O’Ryan, "Virtual Component: a Design Pattern for Memory-Constrained Embedded Applications," *Proceedings of the Ninth Conference on Pattern Language of Programs (PLoP 2002)*, Urbana, IL, USA, 2002.

[CTA+04] D. Conan, C. Taconet, D. Ayed, L. Chateigner, N. Kouici, and G. Bernard, "A Pro-Active Middleware Platform for Mobile Environments," *Proceedings of IASTED International Conference on Software Engineering*, Innsbruck, Austria, 2004.

[CzE00] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*: Addison–Wesley, 2000.

[DeA99] A. K. Dey and G. Abowd, "The Context-Toolkit: Aiding the Development of Context-Aware Applications," *Proceedings of Human Factors in Computing Systems (CHI)*, pp. 434–441, New York, USA, 1999.

[DeK05] L. DeMichiel and M. Keith, "JSR 220: Enterprise JavaBeans, Version 3.0," 2005.

[DENW06] M. Dubinko, I. Expert, A. Navarro, and I. WebGeek, "XHTML™ 2.0," *W3C Working Draft*, 2006.

[Dey01] A. K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing Journal*, vol. 5, pp. 4–7, 2001.

[DKV00] A. V. Deursen, P. Klint, and J. Visser, "Domain-Specific Languages: An Annotated Bibliography," *ACM SIGPLAN Notices*, vol. 35, pp. 26–36, 2000.

[DLB01] P. C. David, T. Ledoux, and N. M. N. Bouraqadi-Saadani, "Two-step weaving with reflection using AspectJ," *Proceedings of OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, Tampa, Florida, 2001.

[Dow98] T. B. Downing, *Java RMI: Remote Method Invocation*: IDG Books Worldwide, Inc. Foster City, California, USA, 1998.

[DPH91] P. Druschel, L. L. Peterson, and N. C. Hutchinson, "Service composition in Lipto," *Proceedings of International Workshop on*

Object Orientation in Operating Systems, pp. 108–111, Palo Alto, California, USA, 1991.

[DSA01] 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, pp. 97–166, 2001.

[DSGO02] F. Doucet, S. Shukla, R. Gupta, and M. Otsuka, "An Environment for Dynamic Component Composition for Efficient Co-Design," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition 2002 (DATE '02)*, pp. 736–743, Paris, France, 2002.

[DSGO03] F. Doucet, S. Shukla, M. Otsuka, and R. Gupta, "BALBOA: A Component-Based Design Environment for System Models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, 2003.

[Duc99] D. Duchamp, "Prefetching hyperlinks," *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS '99)*, Boulder, CO, 1999.

[Emm02] W. Emmerich, "Distributed Component Technologies and their Software Engineering Implications," *Proceedings of the 24th International Conference on Software Engineering*, pp. 537–546, Orlando, Florida, USA, 2002.

[FGC98] A. Fox, S. D. Gribble, and Y. Chawathe, "Adapting to network and client variation using active proxies: Lessons and perspectives," *IEEE Personal Communications*, vol. 5, pp. 10–19, 1998.

[FiS03] D. Fisher and G. Saksena, "Link prefetching in Mozilla: A server-driven approach," *Proceedings of the 8th International Workshop on Web Content Caching and Distribution*, New York, USA, 2003.

[FSLM02] J.-P. Fassino, J.-B. Stefani, J. Lawall, and G. Muller, "THINK: A Software Framework for Component-based Operating System Kernels," *Proceedings of Usenix Annual Technical Conference*, Monterey, USA, 2002.

[FuT99] L. Fuentes and J. M. Troya, "A Java Framework for Web-based Multimedia and Collaborative Applications," *IEEE Internet Computing*, vol. March–April, pp. 56–64, 1999.

[GAO95] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch or why it's hard to build systems out of existing parts," *Proceedings of International Conference on Software Engineering '95*, Seattle, 1995.

[Gar05] J. Garrett, "Ajax: A New Approach to Web Applications," in *www.adaptivepath.com/publications/essays/archives/000385.php*, 2005.

[GCB+00] D. Garti, S.-T. Cohen, A. Barak, A. Keren, and R. Szmit, "Object Mobility for Performance Improvements of Parallel Java Applications," *Parallel and Distributed Computing*, vol. 60, pp. 1311–1324, 2000.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*: Addison–Wesley, 1995.

[Gim02] R. Gimson, "Delivery Context Overview for Device Independence," *W3C Working Draft*, 2002.

[Gim06] R. Gimson, R. Lewis, and S. Sathish, "Delivery Context Overview for Device Independence," *W3C Working Group Note*, 2006.

[Goe02] B. Goetz, "Java theory and practice: Thread pools and work queues – Thread pools help achieve optimum resource utilization," IBM developerWorks, 2002.

[GPSS04] D. Garlan, V. Poladian, B. Schmerl, and J. P. Sousa, "Task-based Self-adaptation," *Proceedings of Workshop on Self-Managed Systems (WOSS'04)*, pp. 54–58, Newport Beach, California, USA, 2004.

[GPZ04] T. Gu, H.K. Pung, and D.Q. Zhang, "Toward an OSGi-based infrastructure for context-aware applications," *IEEE Pervasive Computing*, vol. October–December, pp. 66–74, 2004.

[Gri04] R. Grimm, "One.world: Experiences with a Pervasive Computing Architecture," *IEEE Pervasive Computing*, July–September, pp. 22–30, 2004.

[Gui98] J. de Oliveira Guimaraes, "Reflection for statically typed languages," *Proceedings of 12th European Conference on European Conference on Object-Oriented Programming (ECOOP'98)*, pp. 440–461, Brussels, Belgium, 1998.

[HeF98] J. Helander and A. Forin, "MMLite: A Highly Componentized System Architecture," *Proceedings of 8th ACM SIGOPSE Workshop*, pp. 96–103, Sintra, Portugal, 1998.

[HeI01] K. Henricksen and J. Indulska, "Adapting the Web interface: an adaptive Web browser," *Proceedings of Second Australasian User Interface Conference (AUIC 2001)*, pp. 21–28, Gold Coast, Australia, 2001.

[HHHW97] R. S. Hall, D. Heimbigner, A. van der Hoek, and A. L. Wolf, "An architecture for post-development configuration management in a wide-area network," *Proceedings of 17th International Conference on Distributed Computing Systems*, pp. 269–278, Baltimore, Maryland, 1997.

[HIR01] K. Henricksen, J. Indulska, and A. Rakotonirainy, "Infrastructure for Pervasive Computing: Challenges," *Proceedings of Informatik 2001: Workshop on Pervasive Computing*, pp. 214–222, Vienna, 2001.

[HKSR97] T. D. Hodes, R. H. Katz, E. Servan-Schreiber, and L. Rowe, "Composable ad-hoc mobile services for universal interaction," *Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, pp. 1–12, Budapest, Hungary, 1997.

[HSH+01] T. Haraikawa, T. Sakamoto, T. Hase, T. Mizuno, and A. Togashi, "μVNC: a proposal for Internet connectivity and interconnectivity of home appliances based on remote display framework," *IEEE Transactions on Consumer Electronics*, vol. 47, pp. 512–519, 2001.

[HSP+03] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger, "Context-awareness on mobile devices – the hydrogen approach," *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pp. 292–301, Big Island, Hawaii, 2003.

[ImP07] A. Immonen and M. Palviainen, "Trustworthiness Evaluation and Testing of Open Source Components," *Proceedings of the 7th International Conference on Quality Software (QSIC 2007)*, Portland, Oregon, USA, 2007.

[ITLS04] V. Issarny, F. Tartanoglu, J. Liu, and F. Sailhan, "Software Architecture for Mobile Distributed Computing," *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, Oslo, Norway, 2004.

[Jaa02] A. Jaaksi, "Developing Mobile Browsers in a Product Line," *IEEE Software*, vol. January/August, 2002.

[JHE99] J. Jing, A. S. Helal, and A. Elmagarmid, "Client-Server Computing in Mobile Environments," *ACM Computing Surveys*, vol. 31, pp. 118–157, 1999.

[JJWS98] P. Johnson, H. Johnson, R. Waddington, and A. Shouls, "Task-related knowledge structures: analysis, modelling and

application," *Proceedings of the Fourth Conference of the British Computer Society on People and computers IV*, pp. 35–62, University of Manchester, UK, 1988.

[JoF88] R. E. Johnson and B. Foote, "Designing Reuseable Classes," *Journal of Object-Oriented Programming*, vol. June/July, 1988.

[KAK+00] E. Kaasinen, M. Aaltonen, J. Kolari, S. Melakoski, and T. Laakko, "Two Approaches to bringing Internet services to WAP devices," *Computer Networks*, vol. 33, pp. 231–246, 2000.

[Kat02] E. Katsiri, "Principles of Context Inference," *Proceedings of UbiComp'02*, pp. 33–34, Göteborg, Sweden, 2002.

[KBM+02] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, and H. Morris, "People, Places, Things: Web Presence for the Real World," *Mobile Networks and Applications*, vol. 7, pp. 365–376, 2002.

[KHH+01] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An overview of AspectJ," *Proceedings of the 15th European Conference on Object-Oriented Programming*, pp. 327–353, Budapest, Hungary, 2001.

[KHL02] J. Kimmel, J. Hautanen, and T. Levola, "Display technologies for portable communication devices," *Proceedings of the IEEE*, vol. 90, pp. 581–590, 2002.

[Kic96] G. Kiczales, "Aspect-oriented programming," *ACM Computing Surveys (CSUR)*, vol. 28, 1996.

[Kis07] C. Kiss, "Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 2.0," *W3C Working Draft*, 2007.

[KMF01] E. Kiciman, L. Melloul, and A. Fox, "Towards Zero-Code Service Composition," *Proceedings of the Eighth Workshop on Hot*

Topics in Operating Systems (HotOS VIII), Elmau, Germany, 2001.

[KMSS02] E. P. Kasten, P. K. McKinley, S. M. Sadjadi, and R. E. K. Stirewalt, "Separating introspection and intercession in metamorphic distributed systems," *Proceedings of IEEE Workshop on Aspect-Oriented Programming for Distributed Computing (with ICDCS'02)*, Vienna, Austria, 2002.

[KoC00] F. Kon and R. H. Campbell, "Dependence Management in Component-Based Distributed Systems," *IEEE Concurrency*, vol. 8, pp. 26–36, 2000.

[Kon00] F. Kon, "Automatic Configuration of Component-Based Distributed Systems," in *Department of Computer Science*. Urbana-Champaign: University of Illinois, 2000.

[Kor05] P. Korpipää, "Blackboard-based software framework and tool for mobile device context awareness," *VTT Publications 579*, 2005.

[KPRS03] G. Kappel, B. Pröll, W. Retschitzegger, and W. Schwinger, "Customisation for ubiquitous web applications: a comparison of approaches," *International Journal of Web Engineering and Technology*, vol. 1, 2003.

[KPT94] M. F. Kaashoek, T. Pinckney, and J. A. Tauber, "Dynamic documents: mobile wireless access to the WWW," *Proceedings of Workshop on Mobile Computing Systems and Applications*, pp. 179–184, Santa Cruz, California, USA, 1994.

[KrP88] G. E. Krasner and S. T. Pope, "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System," *Journal of Object-Oriented Programming*, vol. 1, pp. 26–49, 1988.

[KrS93] B. Kroesse and P. Van der Smagt, *An Introduction to Neural Networks*: University Amsterdam Press, 1993.

[KSKB02] J. Kleindienst, L. Seredi, P. Kapanen, and J. Bergman, "CATCH-2004 Multi-Modal Browser: Overview Description with Usability Analysis," *Proceedings of the Fourth IEEE International Conference on Multimodal Interfaces (ICMI'02)*, pp. 442–447, Pittsburgh, Pennsylvania, USA, 2002.

[LaH05] T. Laakko and T. Hiltunen, "Adapting Web content to mobile user agents," *IEEE Internet Computing*, vol. 9, pp. 46–53, 2005.

[LaH05b] O. Layaida and D. Hagimont, "PLASMA: a component-based framework for building self-adaptive multimedia applications," *Proceedings of Electronic Imaging*, pp. 185–196, San Jose, California, USA, 2005.

[LBS+98] J. Loyall, D. Bakken, R. Schantz, J. Zinky, D. Karr, R. Vanegas, and K. Anderson, "QoS aspect languages and their runtime integration," *Proceedings of the 4th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers*, Berlin, Heidelberg, New York, Tokyo, 1998.

[Lew03] R. Lewis, "Authoring Challenges for Device Independence," *W3C Working Group Note*, 2003.

[Lew05] R. Lewis, "Glossary of Terms for Device Independence," *W3C Working Draft*, 2005.

[LiM96] G. Liu and G. Maguire, "A class of mobile motion prediction algorithms for wireless mobile computing and communications," *Mobile Networks and Applications*, vol. 1, pp. 113–121, 1996.

[LKAA96] S. Long, R. Kooper, G. D. Abowd, and C. G. Atkeson, "Rapid Prototyping of Mobile Context-Aware Applications: The Cyberguide Case Study," *Proceedings of the 2nd annual international conference on Mobile computing and networking (MOBICOM '96)*, pp. 96–107, Rye, New York, USA, 1996.

[LMF07] R. Lewis, R. Merrick, and M. Froumentin, “Content Selection for Device Independence (DISelect) 1.0,” *W3C Candidate Recommendation*, 2007.

[LNR96] M. M. Larrondo-Petrie, K. R. Nair, and G. K. Raghavan, "A Domain Analysis of Web Browser Architectures, Languages and Features," *presented at IEEE Southcon’96 Conference*, Orlando, Florida, USA, 1996.

[LuL02] W. Y. Lum and F. C. M. Lau, "A Context-Aware Decision Engine for Content Adaptation," *IEEE Pervasive Computing*, vol. 1, pp. 41–49, 2002.

[Luo98] Luotonen, *Web proxy servers*: Prentice–Hall, Inc. Upper Saddle River, NJ, USA, 1998.

[Mae87] P. Maes, "Concepts and Experiments in Computational Reflection," *Proceedings of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'87)*, pp. 147–155, Orlando, Florida, USA, 1987.

[MaZ04] M. Mamei and F. Zambonelli, "Programming pervasive and mobile computing applications with the TOTA middleware," *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (PerCom 2004)*, pp. 263–273, Orlando, Florida, USA, 2004.

[McA95] J. McAffer, "Meta-level Programming with Coda," *Proceedings of European Conference on Object-Oriented Programming (ECOOP ’95)*, pp. 190–214, Åarhus, Denmark, 1995.

[MCR04] R. Maia, R. Cerqueira, and N. Rodriguez, "An Infrastructure for Development of Dynamically Adaptable Distributed Components," *Proceedings of On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE (OTM 2004)*, pp. 1285–1302, Agia Napa, Cyprus, 2004.

[MDE+95] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer, "Specifying Distributed Software Architectures," *Proceedings of the 5th European Software Engineering Conference*, pp. 137–153, Sitges, Spain, 1995.

[Men00] D. W. Mennie, "An Architecture to Support Dynamic Composition of Service Components and its Applicability to Internet Security," in *Department of Systems and Computer Engineering*. Ottawa: Carleton University, 248 p., 2000.

[MLG01] G. Martin, L. Lavagno, and J. Louis-Guerin, "Embedded UML: a merger of real-time UML and co-design," *Proceedings of International Workshop on Hardware/Software Codesign (CODES 2001)*, Copenhagen, Denmark, 2001.

[Moz98] M.C. Mozer, "The neural network house: An environment that adapts to its inhabitants," *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, pp. 110–114, Menlo Park, California, USA, 1998.

[Moz01] "XPCOM project," Mozilla Organization, 2001.

[MPR01] A. L. Murphy, G. P. Picco, and G. C. Roman, "Lime: A Middleware for Physical and Logical Mobility," *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, pp. 524–533, Phoenix (Mesa), Arizona, USA, 2001.

[MSKC04] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing Adaptive Software," *IEEE Computer*, July, pp. 56–64, 2004.

[MSKC04b] P. K. McKinley, S. M. Sadjadi, E. P. Kasien, and H. C. C. Betty, "A Taxonomy of Compositional Adaptation," Michigan State University, Michigan, USA, 2004.

[MSL99] R. Mohan, J. R. Smith, and C.-S. Li, "Adapting Multimedia Internet Content for Universal Access," *IEEE Transactions on Multimedia*, vol. 1, pp. 104–114, 1999.

[MUCR02] A. L. de Moura, C. Ururahy, R. Cerqueira, and N. Rodriguez, "Dynamic Support for Distributed Auto-Adaptive Applications," *Proceedings of Workshop on Aspect Oriented Programming for Distributed Computing Systems*, pp. 451–456, Vienna, Austria, 2002.

[MuG05] A. Mukhija and M. Glinz, "The CASA Approach to Autonomic Applications," *Proceedings of the 5th IEEE Workshop on Applications and Services in Wireless Networks (ASWN 2005)*, pp. 173–182, Paris, France, 2005.

[NFG06] M. Nussbaumer, P. Freudenstein, and M. Gaedke, "Web Application Development Employing Domain-Specific Languages," *Proceedings of IASTED International Multi-Conference Software Engineering*, Innsbruck, Austria, 2006.

[Nie95] O. Nierstrasz, "Research Topics in Software Composition," *presented at Langages et Modéles á Objets*, 1995.

[NiM94] O. Nierstrasz and T. D. Meijler, "Requirements for a Composition Language," *Proceedings of the ECOOP 94 workshop on Models and Languages for Coordination of Parallelism and Distribution*, pp. 147–161, Bologna, Italy, 1994.

[NiP91] O. Nierstrasz and M. Papathomas, "Towards a type theory for active objects," *Proceedings of the workshop on Object-based concurrent programming*, pp. 89–93, Ottawa, Canada, 1991.

[NiT95] O. M. Nierstrasz and D. C. Tsichritzis, *Object-oriented software composition*: Prentice Hall, 1995.

[Nob00] B. Noble, "System Support for Mobile, Adaptive Applications," *IEEE Personal Communications*, vol. 7, pp. 44–49, 2000.

[NoS95] B. D. Noble and M. Satyanarayanan, "A Research Status Report on Adaptation for Mobile Data Access," *ACM SIGMOD Record*, vol. 24, 1995.

[NoS99] B. D. Noble and M. Satyanarayanan, "Experience with adaptive mobile applications in Odyssey," *Mobile Networks and Applications*, vol. 4, pp. 245–254, 1999.

[NSS01] K. Nagao, Y. Shirai, and K. Squire, "Semantic annotation and transcoding: making Web content more accessible," *IEEE Multimedia*, vol. 8, pp. 69–81, 2001.

[NTMS91] O. Nierstrasz, D. Tsichritzis, V. D. Mey, and M. Stadelman, "Objects + Scripts = Applications," *presented at Esprit 1991 Conference*, Dordrecht, NL, 1991.

[OGT+99] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *Intelligent Systems and Their Applications*, vol. 14, pp. 54–62, 1999.

[OMG96] OMG, "The Common Object Request Broker: Architecture and Specification," Object Management Group, July, 1996.

[OMG02] OMG, "CORBA Components Version 3.0," Object Management Group, June, 2002.

[OMG03] OMG, "Deployment and Configuration of Component-based Distributed Applications Specification," *Draft Adopted Object Management Group Specification*, 2003.

[OMG06] OMG, "CORBA Component Model Specification Version 4.0," 2006.

[Ort03] E. Ortiz, "The MIDP 2.0 Push Registry," January, 2003.

[OSGi03] OSGi, "Open Services Gateway Initiative, OSGi services platform specification," Release 3, *http://www.osgi.org*, March 2003.

[Ous98] J. K. Ousterhout, "Scripting: Higher-Level Programming for the 21st Century," *IEEE Computer*, vol. March, pp. 23–30, 1998.

[Pah02] C. Pahl, "A Formal Composition and Interaction Model for a Web Component Platform," *Proceedings of ICALP'2002 Workshop on Formal Methods and Component Interaction*, Malaga, Spain, 2002.

[PaL03] M. Palviainen and T. Laakko, "mPlaton – Browsing and development platform of mobile applications," *VTT Publications 515*, 2003.

[Pal05] M. Palviainen, "A compositional technique for adapting mobile applications," *Proceedings of the IADIS International Conference WWW/Internet 2005*, pp. 70–75, Lisbon, Portugal, 2005.

[PaL05b] M. Palviainen and T. Laakko, "Using modular and generative approaches for implementing adaptable mobile browser applications," *Proceedings of the IADIS International Conference WWW/Internet 2005*, pp. 101–109, Lisbon, Portugal, 2005.

[PaL05c] M. Palviainen and T. Laakko, ”The construction and integration of XML editor into mobile browser,” *Proceedings of the IS&T/SPIE Electronic Imaging 2005*, vol. 5684, pp. 231–241, San Jose, California, USA, 2005.

[PaL06] M. Palviainen and T. Laakko, "MIMEFrame – A Framework for statically and dynamically composed adaptable mobile browsers," *Proceedings of the 2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, Barcelona, Spain, 2006.

[Pal07] M. Palviainen, "Task-based composition of the context-sensitive UIs of physical environments," *Proceedings of the Third International Conference on Autonomic and Autonomous Systems (ICAS 2007)*, Athens, Greece, 2007.

[Pas97] J. Pascoe, "The Stick-e Note Architecture: Extending the Interface Beyond the User," *Proceedings of International Conference on Intelligent User Interfaces (IUI '97)*, Orlando, Florida, USA, 1997.

[Pas98] J. Pascoe, "Adding Generic Contextual Capabilities to Wearable Computers," *Proceedings of the Second International Symposium on Wearable Computers*, pp. 92–99, Pittsburgh, USA, 1998.

[Pau05] L. D. Paulson, "Building rich web applications with Ajax," *Computer*, vol. 38, pp. 14–17, 2005.

[PBT+04] J. Petzold, F. Bagci, W. Trumler, T. Ungerer, and L. Vintan, "Global State Context Prediction Techniques Applied to a Smart Office Building," *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference*, San Diego, California, USA, 2004.

[PiP99] J. Pitkow and P. Pirolli, "Mining longest repeating subsequences to predict WWW surfing," *Proceedings of the 1999 USENIX Annual Technical Conference*, Monterey, California, USA, 1999.

[PKP03] A. Pashtan, S. Kollipara, and M. Pearce, "Adapting Content for Wireless Web Services," *IEEE Internet Computing*, vol. 7, pp. 79–85, 2003.

[PLF+01] S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd, "ICrafter: A Service Framework for Ubiquitous Computing Environments," *Proceedings of International Conference on Ubiquitous Computing (UBICOMP 2001)*, pp. 56–75, Atlanta, Georgia, USA, 2001.

[PSGS04] V. Poladian, J. P. Sousa, D. Garlan, and M. Shaw, “Dynamic configuration of resource-aware services,” *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*, pp. 604–613, Scotland, UK, 2004.

[Pur94] J. M. Purtilo, "The POLYLITH Software Bus," *ACM Transactions on Programming Languages and Systems*, vol. 16, pp. 151–174, 1994.

[Rab89] L.R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” *Proceedings of the IEEE*, vol. 77, pp. 257–286, 1989.

[RaM04] A. Ranganathan and S. McFaddin, "Using workflows to coordinate Web services in pervasive computing environments," *Proceedings of IEEE International Conference on Web Services* pp. 288–295, San Diego, California, USA, 2004.

[RBH04] P. Rigole, Y. Berbers, and T. Holvoet, "Component-Based Adaptive Tasks Guided by Resource Contracts," *Proceedings of ECOOP Workshop on component-oriented approaches to context-aware systems in conjunction with the 18th European Conference on Object-Oriented Programming*, Oslo, Norway, 2004.

[Rev76] D. Revuz, “Markov chains,” *Bulletin of the American Mathematical Society*, vol. 82, pp. 700–702, 1976.

[RGL98] P.-G. Raverdy, H. L. V. Gong, and R. Lea, "DART: A Reflective Middleware for Adaptive Applications," University of Tsukuba, October, 1998.

[RHBK04] M. Rahnnan, B. Hu, J. Buford, and A. Kaplan, "Mobile multimedia instant messaging and presence services: the architecture and protocols," *Proceedings of IEEE International Symposium on Consumer Electronics*, pp. 208–213, Reading, United Kingdom, 2004.

[RHC+02] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "A Middleware Infrastructure for Active Spaces," *IEEE Pervasive Computing*, vol. October–December, pp. 74–83, 2002.

[Rog97] D. Rogerson, *Inside COM: Microsoft's Component Object Model*: Microsoft Press, 1997.

[RoR05] P. Rossi and C. Ryan, "Empirical Evaluation of Dynamic Local Adaptation for Distributed Mobile Applications," *Proceedings of the International Symposium on Distributed Objects and Applications (DOA 2005)*, pp. 537–546, Larnaca, Cyprus, 2005.

[Ros96] G. van Rossum, "Python Reference Manual," Corporation for National Research Initiatives (CNRI), October, 1996.

[Sam97] J. Sametinger, *Software Engineering with Reusable Components*: Springer, 1997.

[SaM03] S. M. Sadjadi and P. K. McKinley, "A Survey of Adaptive Middleware," East Lansing, Michigan, USA, December, 2003.

[Sat96] M. Satyanarayanan, "Accessing information on demand at any location – Mobile Information Access," *IEEE Personal Communications*, vol. 3, pp. 26–33, 1996.

[SAW94] B. N. Schilit, N. I. Adams, and R. Want, "Context-Aware Computing Applications," *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pp. 85–90, Santa Cruz, California, USA, 1994.

[SBG99] A. Schmidt, M. Beigl, and H. W. Gellersen, "There is more to context than location," *Computers & Graphics*, vol. 23, pp. 893–901, 1999.

[SCH+04] Y. Saida, H. Chishima, S. Hieda, N. Sato, and Y. Nakamoto, "An Extensible Browser Architecture for Mobile Terminals,"

Proceedings of the 24th International Conference on Distributed Computing Systems Workshops (ICDCSW'04), pp. 394–399, Tokyo, Japan, 2004.

[Sch99] J.-G. Schneider, "Components, Scripts, and Glue: A conceptual framework for software composition," 254 p., Bern University, 1999.

[Sch02] D. C. Schmidt, "Middleware for real-time and embedded systems," *Communications of the ACM*, vol. 45, pp. 43–48, 2002.

[Sco00] S. Scott, "Structuring a .NET Application for Easy Deployment," *MSDN Library*, 2000.

[ScT94] B. N. Schilit and M. M. Theimer, "Disseminating Active Map Information to Mobile Hosts," *IEEE Network*, vol. September/October, pp. 22–32, 1994.

[SeA00] M.-T. Segarra and F. André, "A Framework for Dynamic Adaptation in Wireless Environments," *Proceedings of 33th International IEEE Conference on Technology of Object-Oriented Languages and Systems (TOOLS-33)*, pp. 336–347, Mont Saint-Michel, France, 2000.

[Sea02] R. Searls, "Java 2 Enterprise Edition Deployment API 1.1," Sun Microsystems, 2002.

[SEK03] F. J. S. Silva, M. Endler, and F. Kon, "Developing Adaptive Distributed Applications: A Framework Overview and Experimental Results," *Proceedings of the International Symposium on Distributed Objects and Applications (DOA 2003)*, Catania, Sicily, Italy, 2003.

[Ses97] R. Sessions, *COM and DCOM: Microsoft's vision for distributed objects*: John Wiley & Sons, Inc. New York, NY, USA, 1997.

[ShG96] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*: Prentice–Hall, 1996.

[SKK098] T. Souya, M. Kobayashi, S. Kawase, and K. Ohshima, "Joint class experiments based on realtime Web-browser synchronization," *presented at 3rd Asia Pacific Computer Human Interaction*, Shonan Village Center, Japan, 1998.

[SKSK98] H. Sakagami, T. Kamba, A. Sugiura, and Y. Koseki, "Effective Personalization of Push-Type Systems – Visualizing Information Freshness," *WWW7 / Computer Networks*, vol. 30, pp. 53–63, 1998.

[Smi82] B. Smith, "Procedural Reflection in Programming Languages," in *Laboratory of Computer Science*: MIT, 1982.

[Smi06] K. Smith, "Simplifying Ajax-Style Web Development," *IEEE Computer*, vol. 39, pp. 98–101, 2006.

[SMCS04] S. M. Sadjadi, P. K. McKinley, B. H. C. Cheng, and R. E. K. Stirewalt, "TRAP/J: Transparent generation of adaptable Java programs," *Proceedings of the International Symposium on Distributed Objects and Applications (DOA'04)*, Larnaca, Cyprus, 2004.

[SNC00] A. Serra, N. Navarro, and T. Cortes, "DITools: Application-level Support for Dynamic Extension and Flexible Composition," *Proceedings of the USENIX Annual Technical Conference*, pp. 225–238, San Diego, California, USA, 2000.

[SPW02] G. Succi, W.Pedrycz, and R. Wong, "Dynamic Composition of Components using Webcods," *International Journal of Computers and Applications*, vol. 24, 2002.

[Sun02] "Enterprise JavaBeans Specification 2.0," *Sun Microsystems*, 2002.

[SYLZ00] Z. Su, Q. Yang, Y. Lu, and H. Zhang, "What next: A prediction system for web requests using n-gram sequence models," *Proceedings of the First International Conference on Web Information System and Engineering Conference*, pp. 200–207, Hong Kong, 2000.

[VKK01] G. Valetto, G. Kaiser, and G. S. Kc, "A Mobile Agent Approach to Process-based Dynamic Adaptation of Complex Software Systems," *Proceedings of 8th European Workshop on Software Process Technology*, Vienna, Austria, 2001.

[VTA04] N. Venkatasubramanian, C. Talcott, and G. A. Agha, "A Formal Model for Reasoning about Adaptive QoS-Enabled Middleware," *ACM Transactions on Software Engineering and Methodology*, vol. 13, pp. 86–147, 2004.

[VZL+98] R. Vanegas, J. A. Zinky, J. P. Loyall, D. Karr, R. E. Schantz, and D. E. Bakken, "QuO's Runtime Support for Quality of Service in Distributed Objects," *Proceedings of International Conference on Distributed Systems Platforms and Open Distributed Processing*, The Lake District, England, 1998.

[WaG00] Z. Wang and D. Garlan, "Task-Driven Computing," *Carnegie Mellon University CMU-CS-00-154*, 2000.

[WAPP01] "WAP Push Architectural Overview," *WAP Forum*, 2001.

[Wat94] T. Watson, "Application Design for Wireless Computing," *Proceedings of Workshop on Mobile Computing Systems and Applications*, pp. 91–94, Santa Cruz, California, USA, 1994.

[WeB98] G. Welling and B. R. Badrinath, "An architecture for exporting environment awareness to mobile computing applications," *IEEE Transactions on Software Engineering*, vol. 24, pp. 391–400, 1998.

[Wei91] M. Weiser, "The computer for the 21st century," *Scientific American*, pp. 94–104, 1991.

[Wei93] M. Weiser, "Some computer science issues in ubiquitous computing," *Communications of the ACM*, vol. 36, pp. 75–84, 1993.

[WeS00] I. Welch and R. J. Stroud, "Kava – A Reflective Java Based on Bytecode Rewriting," *Reflection and Software Engineering*, vol. 1826, pp. 157–169, 2000.

[WHR+07] K. Waters, R. A. Hosn, D. Raggett, S. Sathish, M. Womer, M. Froumentin, and R. Lewis, "Delivery Context: Client Interfaces (DCCI) 1.0 Accessing Static and Dynamic Delivery Context Properties," *W3C Working Draft*, 2007.

[Win05] D. Winer, "RSS 2.0 Specification," *http://blogs.law.harvard.edu/tech/rss*, 2005.

[XPJ03] X. Xiaoqin, X. Peng, L. Juanzi, and W. Kehong, "A Component Model for Designing Dynamic GUI," *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'03)*, pp. 136–140, Chengdu, China, 2003.

[YeS97] D. M. Yellin and R. E. Strom, "Protocol Specifications and Component Adaptors," *ACM Transactions on Programming Languages*, vol. 19, pp. 292–333, 1997.

[YZL01] Y. Qiang, Z. Haining Henry, and L. Tianyi, "Mining web logs for prediction models in WWW caching and prefetching," *Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining*, San Francisco, California, USA, 2001.

[ZhB04] T. Zhang and B. Brügge, "Empowering the User to Build Smart Home Applications," *Proceedings of 2nd International Conference on Smart homes and health Telematic (ICOST2004)*, Singapore, 2004.

Series title, number and
report code of publication

VTT Publications 658
VTT-PUBS-658

Author(s)
Palviainen, Marko

Title

Technique for dynamic composition of content and context-sensitive mobile applications

Adaptive mobile browsers as a case study

Abstract

The mobile environment brings new challenges for applications. Mobile usage is spontaneous and applications should be fast to install, start, and use in mobile devices and wireless networks. The wireless network connections offer typically less bandwidth than fixed line connections and may cause costs for the user. In addition, the input and output capabilities and memory and processing power resources of mobile devices are typically limited in comparison to desktop computers. This all sets requirements for adaptation methods that could provide more usable and efficient applications for specific users, contexts, and services available on the Web.

Implementation of adaptive applications requires methods for context-sensing and adaptation. The context can change rapidly when a user is moving in a physical environment. Hence, methods that can fast adapt an application for a rapidly changing context are needed. An adaptive application should learn about user behaviour, sense the activity of the user, and use the idle time of the application for speculative adaptation that prepares application parts for potential future contexts in the background. In addition, errors may arise while an adaptive application is being composed for a new context. For example, if a mobile device is disconnected, it is not possible to fetch contents from the Web.

In this dissertation it is argued that the task-based composition technique helps developers to construct adaptive applications for mobile devices and makes the dynamic composition of content and context-sensitive applications more fluent. The task-based composition technique is based on the model of the World Wide Web Consortium (W3C) that provides a requestor-adaptor structure for content adaptation. Like the requestor-adaptor element of the W3C model, a task is an adaptation element that can provide additional context information, request other tasks, adapt their responses, and deliver new or refreshed responses for the requestors. Tasks can prepare content and context-sensitive application instances for current and predicted contexts in many phases and finally compose an application of the prepared instances. The task-based composition technique extends the W3C model with task factories that construct tasks for adaptation requests and specific contexts. Tasks are defined with an XML-based language that enables developers to describe tasks and context-sensitive adaptation actions and their settings. Both context-sensitive tasks and application instances can be cached, which can speed up the adaptation of applications. This dissertation focuses on adaptive browsers that are constructed for mobile devices and discusses how the task-based composition technique can support client-side dynamic composition of content and context-sensitive applications and improve performance when UIs are adapted for rapidly changing contexts and for services available on the Web.

ISBN
978-951-38-7051-5 (soft back ed.)
978-951-38-7052-2 (URL: http://www.vtt.fi/publications/index.jsp)

Series title and ISSN
VTT Publications
1235-0621 (soft back ed.)
1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)

Project number
17670

Date
October 2007

Language
English

Pages
233 p.

Name of project
ALLLAS, WAPproxy, WAP Multimedia, AUTOSPACE, COSI

Commissioned by
VTT, Finnish Funding Agency for Technology and Innovation Tekes, Ulla Tuominen Foundation

Keywords
dynamic composition, task-based composition, mobile application development, adaptive application, adaptive browser, content and context-sensitive application

Publisher
VTT Technical Research Centre of Finland
P.O.Box 1000, FI-02044 VTT, Finland
Phone internat. +358 20 722 4520
http://www.vtt.fi

This dissertation discusses a task-based composition technique that helps developers to construct adaptive applications for mobile devices and makes the dynamic composition of content and context-sensitive applications more fluent. The task-based composition technique is based on the content adaptation model of the World Wide Web Consortium (W3C). Like the requestor-adaptor element of the W3C model, a task is an adaptation element that can provide additional context information, request other tasks, adapt their responses, and deliver new or refreshed responses for the requestors. Tasks can prepare content and context-sensitive application instances for current and predicted contexts in many phases and finally compose an application of the prepared instances. This dissertation focuses on adaptive browsers that are constructed for mobile devices and discusses how the task-based composition technique can support client-side dynamic composition of content and context-sensitive applications and improve performance when UIs are adapted for rapidly changing contexts and services available on the Web.

Julkaisu on saatavana
VTT
PL 1000
02044 VTT
Puh. 020 722 4520
http://www.vtt.fi

Publikationen distribueras av
VTT
PB 1000
02044 VTT
Tel. 020 722 4520
http://www.vtt.fi

This publication is available from
VTT
P.O. Box 1000
FI-02044 VTT, Finland
Phone internat. + 358 20 722 4520
http://www.vtt.fi

ISBN 978-951-38-7051-5 (soft back ed.)
ISSN 1235-0621 (soft back ed.)

ISBN 978-951-38-7052-2 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)