Blended Tailoring in Compositional Web-based Applications

Doctoral Program in Engineering
University of Tsukuba

2003, July

Igor Mejuev

Blended Tailoring in Compositional Web-based Applications

2003, July

Igor Mejuev

A dissertation submitted in partial fulfillment of requirements
for the degree of Doctor of Philosophy in Engineering

Institute of Information Sciences and Electronics
Doctoral Program in Engineering
University of Tsukuba, Japan

Abstract

A *tailorable* computer system allows customization within the context of its use and permits modifications of the program code during its execution. Tailorability allows coping with flexibility requirements, decreasing maintenance cost of software products and accommodating participative software process models that have been applied for the development of interactive Web-based applications. However, the initial cost of developing deeply tailorable systems is considered to be the main reason why tailorability is missing from the majority of currently deployed software products.

This dissertation describes a new methodology of developing component-based applications that allow runtime tailoring. This work envisions the horizontal integration of multiple tailoring interfaces ("blended tailoring") as a way of increasing reusability in the implementation of component-based tailorable systems and correspondingly reducing the applications development costs.

A proposed architectural solution that makes the integration feasible uses nested compositional markup specifications for representing fragments of a tailorable application. This work demonstrates that the decoupling of tailoring interfaces from runtime components allows implementing a generic (application-independent) framework for tailoring that can be utilized in distinct application domains.

The proposed methodology is discussed in the context of a reusable development framework (VEDICI) that has been implemented using Java 2 SDK. The case studies, demonstrating the feasibility of the proposed approach have been carried out using development of a Web-based distance learning application and remote monitoring application in the field of accelerator physics.

Acknowledgements

First of all I would like to sincerely thank Professor Seiichi Nishihara, my adviser, for guiding me through the doctoral program as well as members of the refereeing committee, Professors Hisao Kameda, Nobuo Ohbo, Jiro Tanaka and Yukio Fukui for a number of useful suggestions.

I am also thankful to Professors Yoshihiko Ebihara and Homare Endo of Tsukuba University and to Dr. Akira Kumagai of Tokyo Electron for providing help and support in ways too numerous to mention.

Editors and anonymous referees of journal "Software – Practice and Experience" and IS'2000 conference provided a wide range of constructive comments that contributed to improving the content of this work.

Finally, I would like to thank my family and friends for always cheering me up.

Contents

List of Figures

Chapter 1

Introduction

A *tailorable software system* can continue its evolution after deployment in order to adapt to particular work situation and diverse needs of the users. Tailorability allows coping with flexibility requirements, decreasing maintenance cost of software products and actively involving users on the process of software development and testing. The early and well-known examples of applications incorporating some degree of tailorability are the EMACS editor [Stallman 1981] and Macintosh HyperCard system [Williams 1987].

From the HCI (Human Computer Interaction) perspective, tailoring activity is an activity of modifying a computer application within the context of its use. Tailoring can be also considered as further development of an application during use to adapt it to the requirements that were not accounted for in the original design [Mørch *et al*. 1998].

Tailorability is a natural way to deal with the flexibility requirements and it has a direct impact on the application maintenance cost - an essential property of software products with enterprise deployment. An evaluation [Keen 1991] shows that each dollar spent in the IT industry on new development will yield 0.60¢ of maintenance per year during the lifecycle of the application. However, the increased initial cost of the development of tailorable systems may be the reason why so little tailorability is currently available in products [Appelt *et al*. 1998].

The increased implementation cost is stipulated by low degree of reuse in the implementations of existing tailorable systems. The existing systems also lack generic functionality to be truly effective across multiple application domains.

1.1 Background

The topic of this thesis stems from a research project supported by Telecommunication Advancement Organization of Japan (TAO) on the development of distributed multimedia systems for distance education. The research targeted the development of better tools in support of Web-based remote education and resulted in a preliminary implementation [Mejuev *et al*. 2000; Shimanaka *et al*. 2000; Mejuev *et al*. 2001] of the framework described in the Section 3.6.

The outline of the project is represented in the Figure 1.1. The objective of the system was to support distributed component application building on the Internet. The content developed and refined using the networked clients can be published on a standalone media or executed by portable handheld devices.

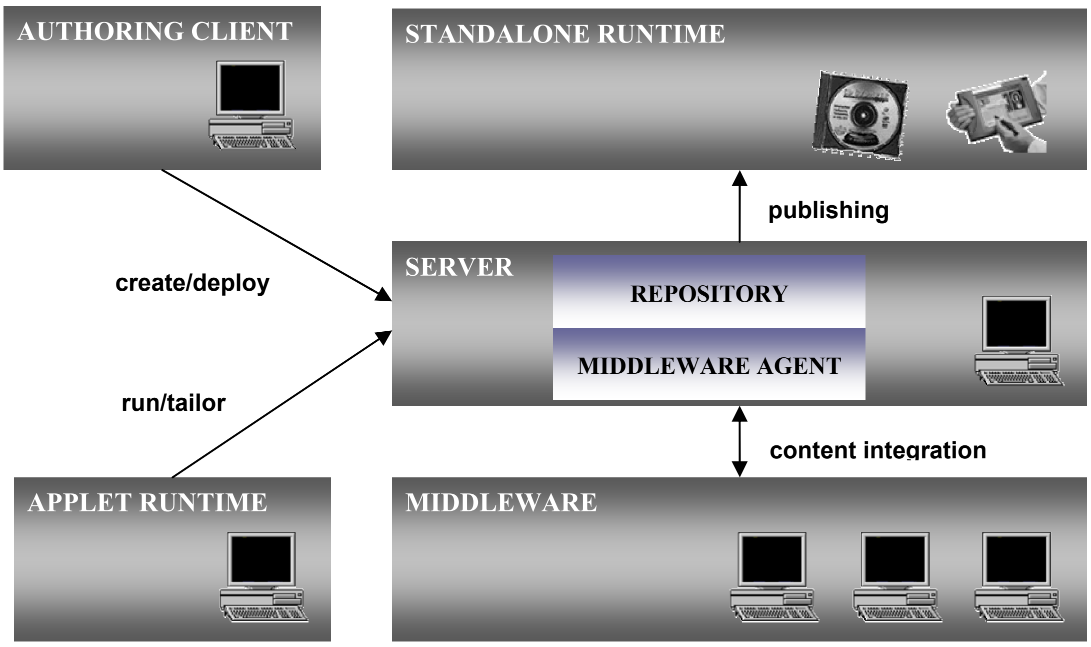


Figure 1.1: "TAO Project" Outline

Later developments on the reusable framework for runtime tailoring allowed extending the application domains to a wider area, such as Scientific and Engineering Computing, taking the development of large-scale control systems in accelerator physics as an example.

1.2 Objectives and Contributions

This dissertation examines tailorable computer systems from software engineering perspectives, the problematic of software development process and the ways of increasing reusability in the implementation of component-based tailorable systems. Additionally, this work proposes an application independent framework for delivering of tailorable Web-based systems with high degree of cross-platform portability. The contributions of this work to the current state of the art in *Software Technology* are as follows:

1. the proposed approach of blended tailoring capable of reducing the development cost of deeply tailorable compositional systems through increasing the reusability of framework modules and software components across the boundaries of proprietary application domains

2. the investigation on software architecture, or the ways of implementing the horizontal integration of multiple tailoring interfaces practically by representing tailorable application fragments with object-scripting technique, based on nested compositional markup specifications

3. the development framework (VEDICI – Visual Environment for DIstributed Content Integration), which allows integrating multiple interfaces within a single application instance and can be used for experiments and exploratory study of efficiency of particular tailoring interfaces

The case studies that have been carried out in this work also made the following contributions to the fields of *Computer Supported Distance Education* as well as *Scientific and Engineering Computing*:

1. an application framework and a set of reusable components for the development of Web-based learning systems, customizable by end-users

2. the consideration of runtime tailorability as a solution for bridging the gap between dynamicity and complexity of requirements in the development of software for large scale control systems in accelerator physics

1.3 Dissertation Structure

Chapter 2 defines basic concepts and terminology related to software process models for Web applications, compositional development, end-user programming, runtime tailoring and compositional markup specification languages.

Chapter 3 introduces the problematic of tailorable software development, develops the concept of blended tailoring and describes an implementation framework (VEDICI) used to verify the proposed approach in practice.

Chapter 4 gives an overview of case studies performed using the development of tailorable remote monitoring and distance learning applications.

Chapter 5 compares VEDICI with existing applications/platforms for runtime tailoring; discusses the results of case studies and usability of the proposed approach in practice.

Chapter 6 presents the conclusions and outlines some perspectives for future work.

Chapter 2

Concepts and Terminology

This chapter defines basic concepts and terminology used in the rest of the thesis. First, the existing classification of *software process models* applicable to the development of Web-based applications is introduced; taking into consideration the problematic associated with each model. Further, the concept of *end-user programming* is discussed as a way to involve the end-users in the software development process. The *runtime tailorability* is considered as a valuable property of software systems that support the paradigm of end-user programming. Finally, the chapter introduces the concept of *compositional development* and describes existing XML-based *compositional markup specification* languages that implement the facilities for object scripting and serialization.

2.1 Software Process Models

Software process models [Sommerville 1996] have arisen in order to bring control to the process of software development. The software process consists of the activities and associated information that are required to develop a software system.

2.1.1 Waterfall

Waterfall model is suitable for large implementation projects where there is a clear goal and software developers or teams work on the system in parallel. The waterfall model partitions the system development into a series of phases and assumes that each phase is completed before the next phase begins. The waterfall declares that the whole system is delivered monolithically at the end of software process lifecycle. The lifecycle phases are defined as follows:

1. Requirements specification
2. Design
3. Implementation and unit testing
4. Integration and system testing
5. Operation and maintenance

The drawback of the waterfall model is the difficulty of accommodating a change after the process is underway. In practice, there is always some interaction (feedback) between phases of the model.

Incremental models are further developments of the waterfall model. Incremental development is the development of a system in a series of partial products that are implemented and delivered to the end-users one by one.

2.1.2 Evolutionary Models

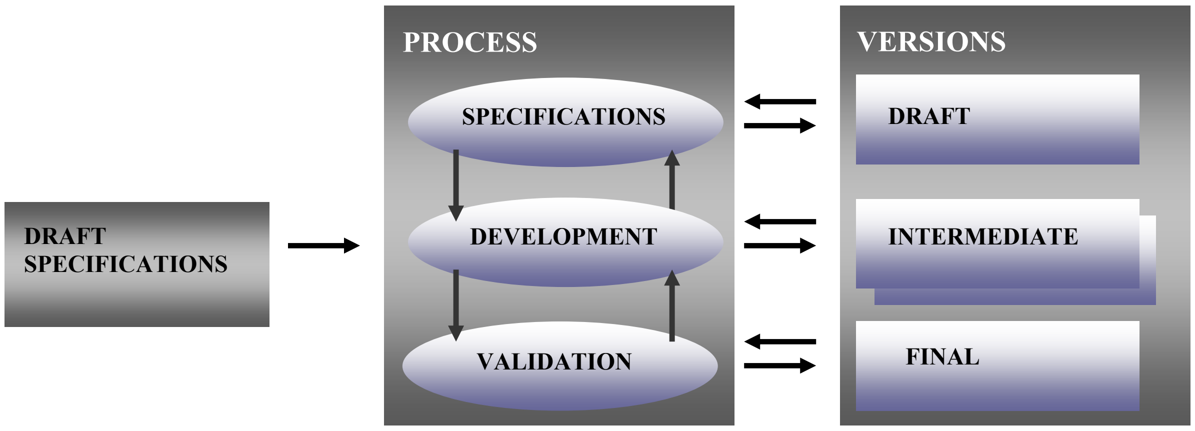


Figure 2.1: Evolutionary Software Process Model

The objective of evolutionary development is to work closely with customers and to evolve a final system from an initial draft specification (Figure 2.1). The initial prototype can be reused in the final version (exploratory prototyping) or its goal can be just the clarification of poorly specified initial requirements (throw-away prototyping).

The strategy of evolutionary development is to deliver a preliminary version of the system to the end-users as soon as possible. Further, the software process is going through a series of intermediate versions, evaluated and validated jointly with end-users. The *participative* model of evolutionary software design [Floyd *et al*. 1989] is developed around the concept that the requirements are not given and therefore are established gradually through interaction between users and developers.

Taking into account the highly interactive and dynamic nature of applications for the Web, the choice of appropriate software process models for these applications often leads to evolutionary and participative models. For the purposes of this work, an application is considered to be Web-based if it relies on URL-addressable resources and may be accessed via a Web-browser providing integration with built-in browser facilities. Web-based applications play an increasingly important role in software technology related to online services, internet portals, online monitoring, personal information retrieval and storage systems.

The problems of evolutionary development model that affect Web-based development include lack of overall process visibility, poor software structuring and reusability, as well as the need for particular skills to be acquired by developers, such as the knowledge of special languages for fast prototyping. In particular, the problem of additional development skills underlines the need to study system-analysis approaches and implementation techniques that allow acquiring and incorporating the evolving requirements directly from the end-users.

2.2 End-User Programming

At the development stage it is extremely difficult to take into account all the details of the tasks performed by end-users. In an attempt to cover a wide audience the programs are often packaged with hundreds of features users never know and use. In this context providing the possibility for end-user customization could make the use and development of software much simpler.

However, practical implementation of such a possibility is not trivial. The system design must permit the acceptance of customizations coming from end-users in appropriate places. There must be a way to support differentiation, persistence and management of the applied changes. The expressiveness and generality of traditional programming languages should be shaded by appropriate metaphors capable of increasing the usability. The metaphors can utilize a variety of techniques, such as learning-by-example, visual programming or domain-specific approaches [Cypher 1993].

2.3 Runtime Tailorability

Runtime tailorability allows incorporating the end-user programming techniques into a running application. User interface tailorability has been extensively researched from HCI perspectives. The research established categorizations of tailorability functions, classification of users that would use tailorability tools and produced a number of research prototypes.

Different *levels* of tailorability can be distinguished, corresponding to the different aspects of a computer application [Mørch 1997]:

1. *customization:* manipulating switches
2. *integration:* changing application composition
3. *extension:* changing underlying implementation code

The following classification of the end-users doing tailoring has already been established; [MacLean *et al.* 1990] this classification takes into account the level of understanding of a system of a given user, rather than a user's level of computing skills:

1. *workers*: just need some work to be done, have no expectations of tailoring functionality,
2. *tinkerers*: power users, want to explore tailoring functions, but may lack understanding of the system,

3. *programmers*: can specify production rules or object classes, fully understand the system they use.

There is also a corresponding consideration and experiment on "tailoring culture" where users can feel in control of a system and in which tailoring is a norm [MacLean *et al.* 1990].

Research prototypes targeting tailorability in a particular application domain range from desktop applications [MacLean *et al.* 1990] to collaborative Web workspaces [Appelt *et al.* 1998]. There exist increasing interest in implementing tailorability for CSCW (Computer Supported Cooperative Work) applications [Mørch *et al.* 1998], including the ongoing development of the generic tailoring platform for CSCW [Stiemerling *et al.* 1999]. The syntax of CAT-files (Component Architecture for Tailoring) – part of EVOLVE platform [Stiemerling *et al.* 1999] can be considered as an example of developments related to proprietary specification languages for tailorable applications.

2.4 Compositional Markup Specifications

Compositional markup specification (CMS) languages use markup syntax, such as XML [W3C 2000] to represent the composition and links among components of an application. The use of XML as a foundation for defining scripting languages is motivated by the availability of standardized APIs [W3C 1998; Migginson and Brownell 2002], interfaces and tools to process XML documents. Moreover, the integration of XML into popular frameworks such as Java SDK made it possible to reuse the scripting techniques for long-term serialization of object states.

2.4.1 Compositional Development

The construction of compositional applications from reusable components, integrated by means of scripting, can resolve some deficiencies in applying object-oriented languages to component-based development. Defining rich public interfaces in OOPL makes the composition more difficult; moreover object-oriented languages typically provide a very limited binding technology for composing software – it is necessary to program new objects to integrate the existing ones.

In general, component-scripting technology allows shifting from individual and monolithic application development to the development of standard components, interfaces and tools [Nierstrasz *et al*. 1991].

2.4.2 Markup Languages

Current examples of compositional markup languages are introduced below.

BML (Bean Markup Language) – a wiring XML-based language for Java Beans, which directly represents the Java Beans' component model. BML includes a language grammar specification, represented by XML Document Type Definition (DTD) as well as API that is used to process XML documents composed in accordance with BML DTD [Johnson 1999]. The design goals of BML include offering full support for Java Beans' specifications and providing means for configuring arbitrary bean sets.

The *java.beans.XMLDecoder* class [Sun 2002] has been released with Java SDK 1.4 as an improvement of the built-in Java serialization API. The implementation is based on the "archives are programs" concept that replaces recording an object's internal state with reconstituting the state using public APIs (archiving vs. marshaling). This approach incorporates a mechanism which eliminates redundant statements and improves serializing performance and fault-tolerance [Milne 1999]. Java 1.4 also provides the corresponding *java.beans.XMLEncoder* class that can be used to create a textual, XML-based representation of an arbitrary graph of JavaBeans.

While representing compositional languages for beans scripting, the above languages "as is" do not allow delivery of a controllable degree of granularity (nesting) and support for visualization tools. These are essential requirements for specifying the composition of tailorable applications.

Chapter 3

Blended Tailoring

This chapter introduces a methodology and implementation framework, which can be reused as a foundation for developing a variety of component-based tailorable applications.

First, the chapter considers the scope the proposed approach, summarizes the differences between tailoring and authoring interfaces and the problematic of developing tailorable applications for the Web. Further, the detailed consideration is presented on the methodology of developing compositional tailorable applications (*blended tailoring)* and implementation platform (VEDICI), which proves the feasibility of the proposed methodology.

3.1 Scope

Taking into account the highly interactive and dynamic nature of modern applications for the Web, the choice of appropriate software process for these applications naturally leads to evolutionary and participative models. The same consideration is true for development of tailorable Web-based applications.

This work assumes that the increment of the evolutionary development process for a tailorable Web application comprises the following steps:

1. Identifying the necessary *degree of tailorability* by summarizing the assumptions about the needs of intended users' community.

2. Identifying the *architectural solution* or the way of implementing the necessary degree of tailorability.

3. Performing a *usability inspection* of deployed system.

The careful selection of test users involved into design process is considered to be the key issue in capturing the diversity and evolution of requirements and in identifying the required degree of tailorability [Stiemerling *et al.* 1997].

This work addresses the issue (2) and focuses on *component-based* systems that implement tailorability "by integration". It is further assumed that component-based tailoring activity is performed by configuring the properties of existing components, incorporating new components, or assembling the components in a new way.

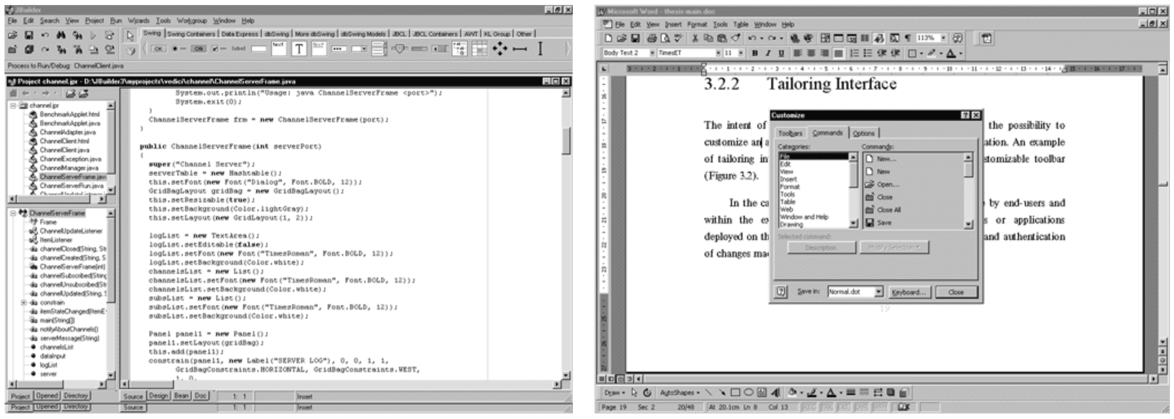
The building block examples include objects, views, agents and links in OVAL [Malone *et al.* 1992], information, collaboration and interface objects in ICE [Farshchian 1998], and FLEXIBEANS in EVOLVE [Stiemerling *et al.* 1999]. A user can manipulate a composition of building blocks by means of generic visual programming techniques such as form-based programming, data-flow programming, or use a proprietary technique for a given application domain.

The need to place *tailoring interfaces* into a distinct subset of user interfaces (UI) is motivated by the fact that deep tailoring may affect the presentation and/or business logic of a running application. The users need to be aware of this mode of operation since it requires a distinct cognitive view of a task being performed.

This dissertation suggests that the high initial development cost of end-user tailorable systems originates in low degree of reusability, particularly as applied to the UI that a system offers to the end-users for doing tailoring. Tailorable applications are often seen as domain-specific environments for *authoring* (end-user programming, rapid application development) - a view that does not admit a system development methodology perspective.

3.2 Authoring versus Runtime Tailoring

User interfaces for runtime tailoring can visually look very similar to the ones used for authoring, however there are important differences that should be taken into consideration in order to distinguish between software designs for software development and for runtime tailoring.



```
        System.out.println("Usage: java ChannelServerFrame <port>");
        System.exit(0);
    }
    ChannelServerFrame frm = new ChannelServerFrame(port);
}

public ChannelServerFrame(int serverPort)
{
    super("Channel Server");
    serverTable = new Hashtable();
    this.setFont(new Font("Dialog", Font.BOLD, 12));
    GridBagLayout gridBag = new GridBagLayout();
    this.setResizable(true);
    this.setBackground(Color.lightGray);
    this.setLayout(new GridLayout(1, 2));

    logList = new TextArea();
    logList.setEditable(false);
    logList.setFont(new Font("TimesRoman", Font.BOLD, 12));
    logList.setBackground(Color.white);
    channelsList = new List();
    channelsList.setFont(new Font("TimesRoman", Font.BOLD, 12));
    channelsList.setBackground(Color.white);
    subsList = new List();
    subsList.setFont(new Font("TimesRoman", Font.BOLD, 12));
    subsList.setBackground(Color.white);

    Panel panel1 = new Panel();
    panel1.setLayout(gridBag);
    this.add(panel1);
    constrain(panel1, new Label("SERVER LOG"), 0, 0, 1, 1,
        GridBagConstraints.HORIZONTAL, GridBagConstraints.WEST,
        1, 0,
```

3.2.2 Tailoring Interface

Figure 3.1: Authoring and tailoring interfaces.

An authoring interface, such as Integrated Development Environment (Figure 3.1, left) is typically employed by developers of a software system. The interface can utilize techniques such as visual programming or form-based programming, in order to speed up the process of development. The authoring system is required to provide full control of the application and available APIs, display the composition of the system in a consistent way, and provide integration with runtime and deployment modules. The software process comprises iterative steps such as deploying the application's template, identifying the required modifications through the communication with users, modifying the application, deploying the new version, and so on. Typically the users and developers of the system represent distinct and geographically distributed groups.

Contrary to the above, the purpose of a *tailoring* interface is to enable users to customize an application while it is running. An example of tailoring interface is a text processing application with customizable toolbar (Figure 3.1, right). In the case of tailoring, modifications are performed by end-users and within the execution environment. For shared applications or applications deployed on the Web, the system should support persistence and authentication of changes made by each user.

The incorporation of tailoring interfaces at runtime imposes an additional complexity on the implementation of a system that supports runtime customizations. The implementation, therefore, should rely on corresponding techniques such as computational reflection.

3.3 The Problematic

The main problems faced by a designer of a compositional tailorable system for the Web identified in this work are as follows:

1. Selection of appropriate application decomposition and visualization techniques can not be fixed at the design phase without imposing limitations on the usability of the final product. In general, the proper choice of components and visualization techniques is domain-dependent and user requirements and preferences regarding them continue evolving throughout the project lifecycle.

2. Complex applications do not anticipate a ″universal″ style of tailoring interfaces. Building from a ″minimal″ set of components may be sufficient for developers, but they might be rejected by end-users or lead to increased learning costs. Moreover, complex applications require a mixture of tailoring interfaces to be supported. For example, a front-end of a data analysis application would require one visualization style for tailoring the visual preferences of the UI (e.g. property sheets) but a different style for tailoring the data-processing logic (e.g. data-flow).

In order to address the issues of implementing tailorable applications for the Web, this work proposes shifting from a fixed specification of how the tailoring should be performed to a flexible specification, which allows delivering end-user systems without having to ″freeze″ the UI for tailoring in the design phase. Moreover, the integration of multiple tailoring interfaces can be implemented "horizontally" - a proposed framework allows not only switching tailoring interfaces to match the level of a user (worker, tinkerer, programmer), but also employing multiple tailoring UIs simultaneously, in order to visualize different aspects of an application with appropriate tailoring interfaces (Figure 3.2).

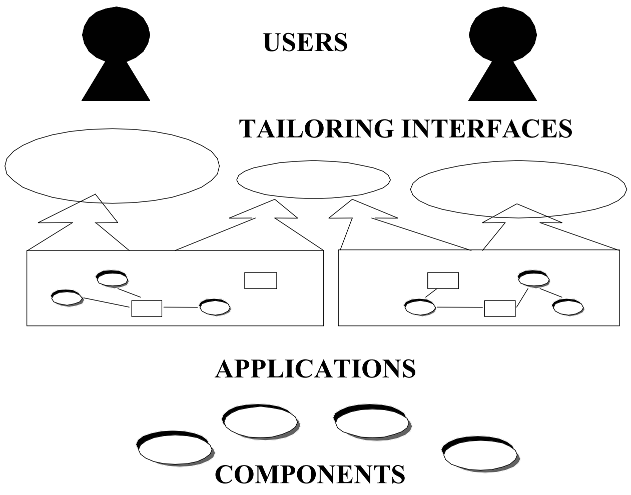


Figure 3.2: Blended tailoring.

3.4 Modules of a Generic Tailoring Framework

The primary purpose of a *tailoring framework* is to provide a set of tools and libraries for software developers, however, applications usable for all categories of end-users can be based on it [Mørch 1997]. A framework that implements compositional tailoring and allows the delivery of systems in real-world-application domains needs to provide the following infrastructure:

1. Integrating *runtime* that would enable dynamic recomposing of applications and support for multiple tailoring interfaces.

2. APIs for *de-serializing composite components* that runtime employs. The serialized presentation can range from high-level compositional languages to binary serialization in an extreme case.

3. Persistence of customized applications in a Web-based system requires the implementation of an application *repository* with authorization and access control in order to differentiate the changes applied to the applications by end-users.

4. Application-specific or reusable primitive *components* and concrete implementations of *tailoring interfaces*.

Application repository, runtime and de-serializing APIs are application-independent elements of the above infrastructure and they can be used to form a generic framework for the development of tailorable Web-based applications.

3.5 The "Visualizer" Pattern

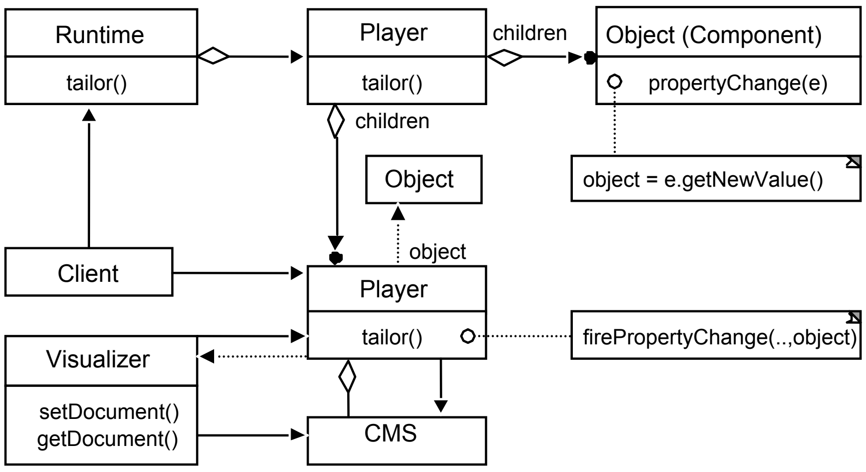


Figure 3.3: The "visualizer" pattern.

Architecture for recomposing applications at runtime is presented in Figure 3.3 using a modified OMT notation introduced in [Gamma *et al.* 1995]. In terms of modules of a generic framework for tailoring, this figure describes "integrating runtime" that utilizes compositional markup specifications (CMSs) as serialized presentations of composite components. The runtime employs a nested hierarchy of wrappers for composite components (*players*) with the ability to associate a tailoring entity (*visualizer*) with each player. A player holds a CMS, which defines a composite component and recompiles the component if any modifications on the CMS are made by its associated visualizer. Mappings between players and visualizers are assigned as modifiable properties of players enabling support for multiple tailoring interfaces per application and, moreover, enabling the visualizers to be dynamically reassigned at runtime, if necessary.

Tailoring is initiated by a component of an application or by a framework by dispatching the corresponding event (″tailor″) to a player. Having received the ″tailor″ event, a player instantiates its associated visualizer parameterized with the CMS held by the player. The visualizer provides a user interface for CMS authoring and asynchronously returns the updated CMS back to the player. The player recompiles its composite component creating a corresponding application object and uses a standard JavaBeans notification mechanism by firing a ″propertyChange″ event for its bound ″object″ property. The ″propertyChange″ event can be caught by components in the parent player and trigger application-dependent actions, such as repainting of the UI. To prevent overlapping updates, the concurrent opening of multiple visualizers for a player is not allowed. An object reference from visualizer to its player allows implementing ″change locks″ and provides visualizer with the ability to access the current properties of components in a player verifying consistency of their composition, if necessary. The described flow of events can be controlled programmatically by application developers or, it can be executed by framework, propagating the ″tailor″ event through a root player to a player that needs to be tailored. Since this scenario requires recompiling only one nested composite component at a time, the states of other components are preserved, allowing an application to be modified while it is still running and for a pool of visualizers to be applied, which can provide distinct styles of tailoring interfaces at the appropriate places.

3.6 Framework Implementation

VEDICI is an implementation of the architecture described in the previous sections, using the Java 2 Platform[1]. VEDICI is intended for Java developers who need a framework for delivering end-user tailorable Web-based applications. An outline of this environment is presented in Figure 3.4. Following is a description of VEDICI modules and their interactions.

[1] The total metrics of source code for reusable system modules: LOC ≈ 11500, McCabe Cyclomatic Complexity (logical branching) ≈ 1700.

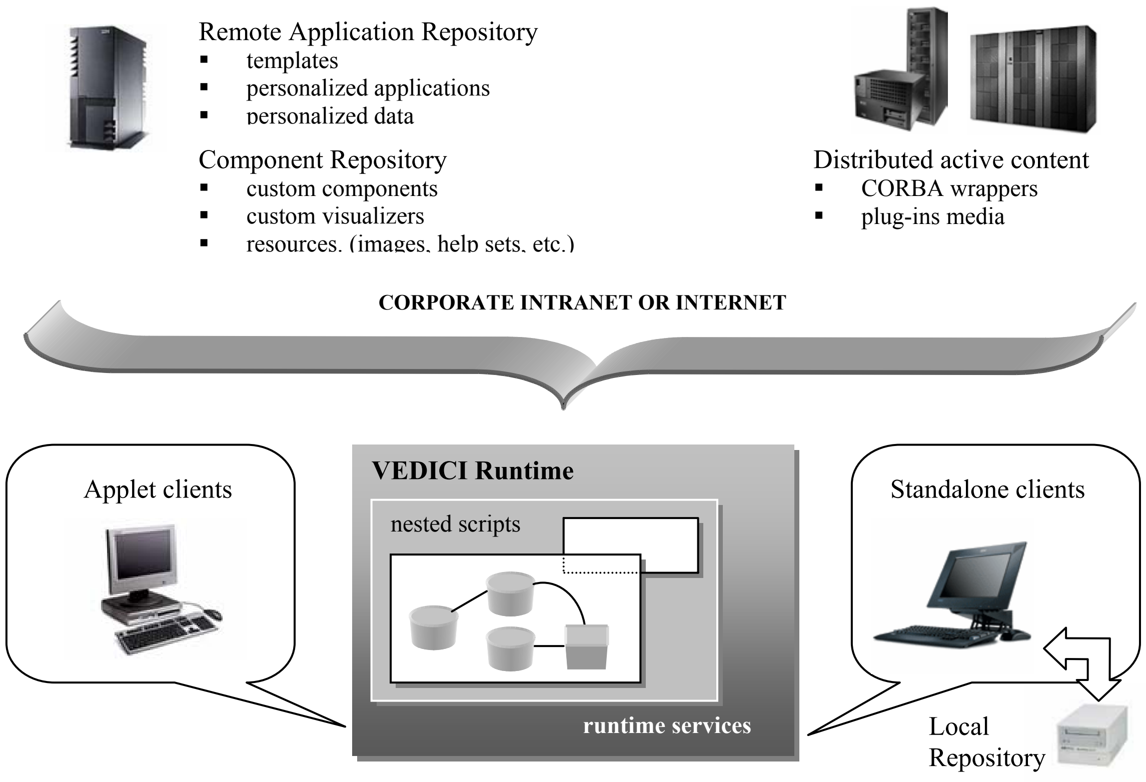


Figure 3.4: Composition of VEDICI.

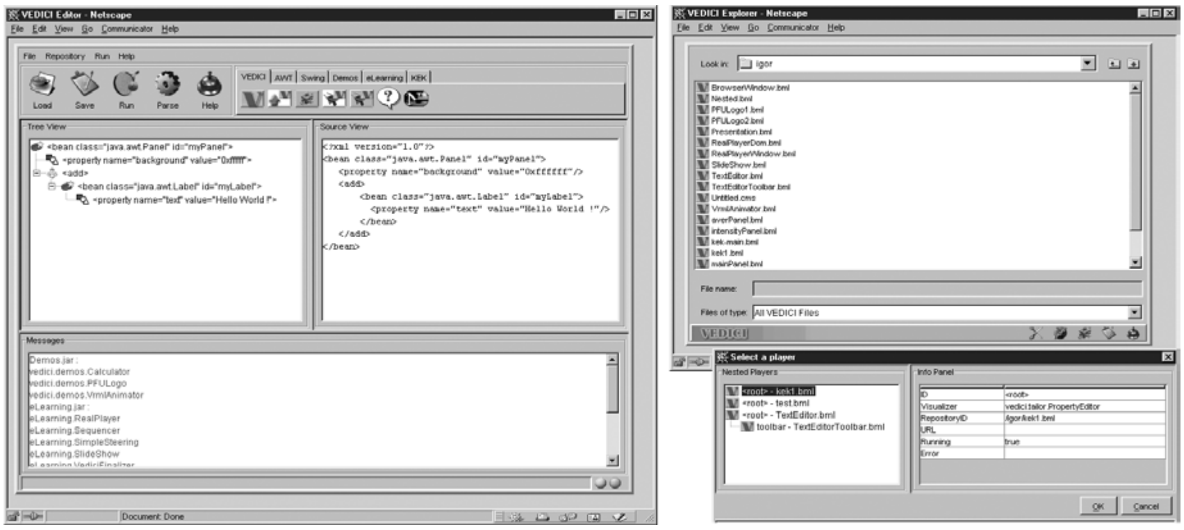
3.6.1 VEDICI Runtime

VEDICI Runtime is an UI front-end that implements the ability to dynamically assemble a tailorable Web-based application from primitive components (JavaBeans), visualizers and nested compositional markup scripts. The runtime is implemented to run either as a standalone application or with a Java plug-in supported by all major browsers and platforms.

The runtime uses reflection (*java.lang.reflect* package) and a BML API that has been extended with support for applications-nesting, asynchronous data exchange between applications running in parallel and nesting and support for the application life cycle. BML has been selected among other toolkits available for processing of CMSs, primarily due to its robustness. Primitive application components can access APIs, provided by the environment to use the facilities of the client's Web-browser (e.g. interfacing with Netscape plug-ins or OCX controls in MSIE) and, repository services (see below) in order to save or load personalized information, such as configuration data, for the current user. The application components can optionally access CORBA interfaces through the Gatekeeper, provided by VisiBroker ORB.

VEDICI Runtime includes a set of documented abstract classes and interfaces to support the development of custom visualizers, application repositories and primitive components that make use of runtime APIs. The runtime UI allows execution and tailoring of a single application loaded from a URL or application repository. The UI and user interaction scenario for VEDICI Runtime is described in greater details in the section 3.6.4 below.

In order to facilitate framework utilization by end-users with various levels of skills and work-style preferences, two alternative UI front-ends are available (Figure 3.5) in addition to the VEDICI Runtime (Figure 3.7):



```
<bean class="java.awt.Panel" id="myPanel">
  <property name="background" value="0xffffff">
  <add>
    <bean class="java.awt.Label" id="myLabel">
      <property name="text" value="Hello World !">
```

```
<?xml version="1.0"?>
<bean class="java.awt.Panel" id="myPanel">
   <property name="background" value="0xffffff"/>
   <add>
       <bean class="java.awt.Label" id="myLabel">
         <property name="text" value="Hello World !"/>
       </bean>
   </add>
</bean>
```

```
Demos.jar :
vedici.demos.Calculator
vedici.demos.PFULogo
vedici.demos.VrmlAnimator
eLearning.jar :
eLearning.RealPlayer
eLearning.Sequencer
eLearning.SimpleSteering
eLearning.SlideShow
eLearning.VediciFinalizer
```

```
BrowserWindow.bml
Nested.bml
PFULogo1.bml
PFULogo2.bml
Presentation.bml
RealPlayerDom.bml
RealPlayerWindow.bml
SlideShow.bml
TextEditor.bml
TextEditorToolbar.bml
Untitled.cms
VrmlAnimator.bml
averPanel.bml
intensityPanel.bml
kek-main.bml
kek1.bml
mainPanel.bml
```

```
<root> - kek1.bml
<root> - test.bml
<root> - TextEditor.bml
    toolbar - TextEditorToolbar.bml
```

ID	<root>
Visualizer	vedici.tailor.PropertyEditor
RepositoryID	/igor/kek1.bml
URL	
Running	true
Error	

Figure 3.5: VEDICI Editor and Repository Explorer.

1. *VEDICI Editor* is an authoring tool with the look and feel of a standard IDE which allows loading/saving application scripts from/to a repository, editing the script sources, and application execution and tailoring.

2. *VEDICI Explorer* implements an "explorer" UI that allows browsing the content of an application repository as well as application execution and tailoring.

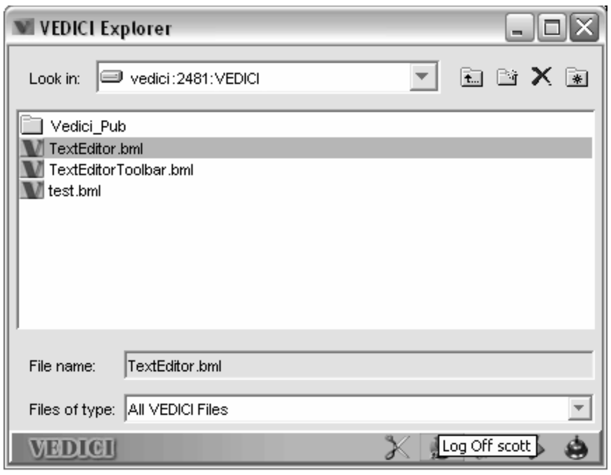
3.6.2 Application Repository

The scripts can be loaded from and saved to an *Application Repository*. The framework includes the following repository classes that can be employed at the discretion of developers:

1. *CGIRepository* – provides remote interfaces to a Web-server file system. Server-side of this repository is implemented as a Perl/CGI script. CGI repository access requires an authorization by username and password that allows differentiating the changes applied to the scripts by the end-users.

2. *LocalFileReposiory* – interfaces directly with a local file system on a client's computer and is suitable for stand-alone, non-shared applications.

3. Integration of a *custom repository* is also supported. Support for custom repositories is implemented using the "Strategy" pattern [Gamma *et al*. 1995]. The name of a Java class providing custom repository implementation may be specified either as an applet parameter or as a command line option. Additionally, initial versions of the scripts (templates) can be loaded from a given URL.



Look in: vedici:2481:VEDICI

Vedici_Pub
TextEditor.bml
TextEditorToolbar.bml
test.bml

File name: TextEditor.bml

Files of type: All VEDICI Files

Figure 3.6: Explorer with EJB/Oracle repository.

An example of utilizing the custom repository facility is *EJBRepository* (Figure 3.6) which emulates a virtual file system interface using relational database tables. Server-side of this repository is implemented as an EJB running in Oracle JServer/Aurora container; it uses Oracle database accounts for authorization. This design allows utilizing obligatory encryption of user passwords and optional encryption of all the information transferred over a network. EJBRepository also implements a virtual, read-only folder which is accessible to all of the repository users. In Oracle terminology the shared folder is represented by a "public synonym" (Vedici_Pub) which is owned by an "administrative account" schema. Correspondingly, only the administrative account is allowed to modify the shared files.

3.6.3 Component Repository

Component Repository holds a collection of primitive components of VEDICI applications. The repository contains a set of archive files in JAR format. Each JAR file includes a set of logically related Java Beans together with all resources required by beans. In implementation terms, the component repository interface allows VEDICI Runtime to dynamically form a Java "class loader" which enables referencing all of the repository beans from nested CMSs.

3.6.4 Sample Application

A sample application demonstrates a basic use of the "visualizer" pattern introduced in Figure 3.3. This application implements a text editor with customizable toolbar. The composition and a screenshot of the sample application are presented in Figure 3.7.

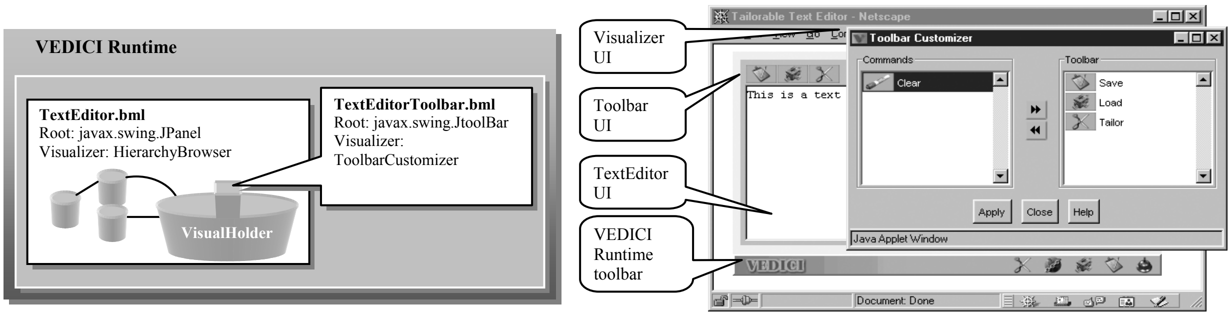


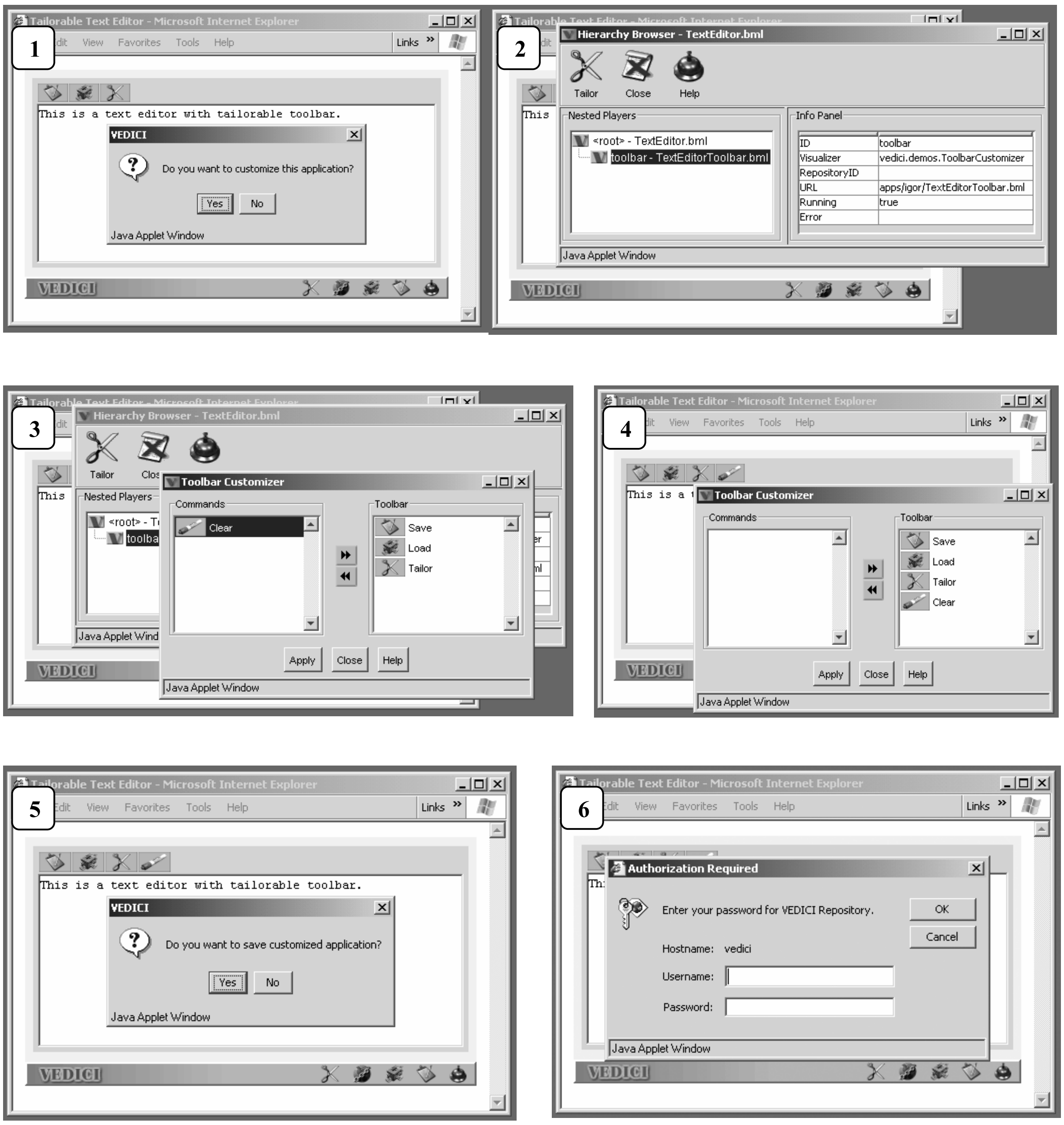
Figure 3.7: The composition and a screenshot of a sample application.

The root application object (a *javax.swing.JPanel* instance) is defined by TextEditor.bml script that uses a nested toolbar definition (TextEditorToolbar.bml) to obtain a toolbar object (*javax.swing.JToolBar* instance). A simplified outline of the TextEditor.bml markup is presented below:

```
<?xml version="1.0"?>
<bean class="javax.swing.JPanel" id="frame">
    ...
    <bean class="vedici.runtime.VediciPlayer" id="toolbar"/>
      <property name="url" value="apps/igor/TextEditorToolbar.bml"/>
      <property name="visualizer"
        value="vedici.demos.ToolbarCustomizer"/>
    ...
    <bean class="vedici.tailor.VisualHolder" id="holder">
      <property name="player">
         <bean source="toolbar"/>
      </property>
    </bean>
    ...
    <bean class="vedici.runtime.VediciPlayerStub" id="stub">
      <property name="visualizer"
        value="vedici.tailor.HierarchyBrowser"/>
    </bean>
    ...
    <bean class="javax.swing.JTextArea" id="textArea"/>
    ...
    <add>
      <bean source="textArea"/>
      <string value="Center"/>
    </add>
    <add>
      <bean source="holder"/>
      <string value="North"/>
    </add>
...
</bean>
```

A nested TextEditorToolbar.bml script is loaded by a *vedici.runtime.VediciPlayer* instance and assigned an application-specific visualizer (Toolbar Customizer). The toolbar is wrapped into an instance of *vedici.tailor.VisualHolder*, which can handle the differences in visual component rendering between Java AWT and Swing UI libraries and which can also implement dynamic UI-recomposing in a library-independent fashion.

The parent application (TextEditor.bml) is assigned a reusable visualizer (Hierarchy Browser), which allows browsing the structure of an application and calling nested visualizers for tailoring particular aspects of the application. The association is done via a *vedici.runtime.VediciPlayerStub* instance that permits accessing the properties of a player that is compiling a script from within the script being compiled. The rest of the TextEditor.bml script defines additional application components and UI layouts in a container. A customization scenario for the sample application is presented in Figure 3.8:



This is a text editor with tailorable toolbar.

Do you want to customize this application?

Yes No

<root> - TextEditor.bml
toolbar - TextEditorToolbar.bml

ID	toolbar
Visualizer	vedici.demos.ToolbarCustomizer
RepositoryID	
URL	apps/igor/TextEditorToolbar.bml
Running	true
Error	

Do you want to save customized application?

Yes No

Enter your password for VEDICI Repository.

Hostname: vedici

Username:

Password:

Figure 3.8: Customization scenario for a sample application.

1. An end-user initiates an application customization via a "Tailor" action on the VEDICI Runtime toolbar. A confirmation dialog "Do you want to customize this application?" is shown, the user clicks "Yes".

2. A visualizer for the root application player is instantiated (Hierarchy Browser). Using the Hierarchy Browser UI the user selects a nested toolbar player and initiates tailoring for that player.

3. A visualizer for toolbar player is instantiated (Toolbar Customizer).

4. Using the Toolbar Customizer UI, the user adds a "Clear" button to the text editor toolbar.

5. The user closes visualizers and initiates the "Save" action on the VEDICI Runtime toolbar. A confirmation dialog "Do you want to save customized application?" is shown, the user clicks "Yes".

6. An authorization dialog for the VEDICI repository is shown.

Once the authorization is complete, a customized version of the whole application is saved to the user's personal storage space in the VEDICI repository. Later, the user can load the customized application via the "Load" action on the VEDICI Runtime toolbar.

In addition to the above, the TextEditorToolbar.bml script implements a "Tailor" button, which invokes the "tailor" method of the toolbar player and directly takes the user to step (3) in the above scenario, without displaying the UI of the Hierarchy Browser:

```
<?xml version="1.0"?>
<bean class="javax.swing.JToolBar" id="tb">
  ...
  <bean class="vedici.runtime.VediciPlayerStub" id="stub"/>
  ...
  <add>
    <bean class="javax.swing.JButton" id="tailor">
      ...
      <event-binding name="action">
        <script>
          <call-method target="stub" name="tailor"/>
        </script
      </event-binding>
    </bean>
  </add>
</bean>
```

Cross-referencing between scripts located in users' personal folders provides the possibility for implementing centralized tailoring. For instance, the TextEditor.bml script loads the toolbar definition from the URL "apps/igor/TextEditorToolbar.bml" - this URL corresponds to a user's personal folder managed by remote repository and is simultaneously accessible via a Web server. If a folder's owner makes changes to the toolbar script, it will affect initial versions (templates) of the toolbar application for all other users.

It should be noted that the Text Editor application itself does not include any facilities for toolbar customization; instead, it relies on a reusable framework, which can potentially provide the tailoring facilities needed for arbitrary component-based applications and support persistence of changes made by each user.

3.7 Summary

This chapter described a methodology of implementing component-based applications that can be tailored at runtime and a generic framework intended for developers of customizable Web-based applications. A simple implementation of a tailorable text editor has been presented in order to illustrate the framework design and its interaction with the end-users. The following chapter introduces case studies on the practical applications of the proposed framework.

Chapter 4

Case Studies

This chapter describes case studies on the practical applications of the proposed framework that was introduced above. Contrary to the sample application described in the previous chapter, applications in real-world domains require a higher degree of nesting as well as a wider selection of reusable and application-dependent components, visualizers and scripts. In general, the scope of tailoring does not have to be limited to the user interface customization, e.g. the data-processing logic can be customized in the same way. Moreover, a visualizer instance can interface with the end-user via a series of dialogs, without employing a distinct window to handle all of the interaction.

4.1 Remote Monitoring Application

4.1.1 Background

The application of runtime tailoring can solve the contradiction between dynamicity of requirements and inherent complexity of software present in some application domains.

An accelerator control system is an example of such a domain – the dynamicity and flexibility are the essential requirements for scientific experiment environment, however the amount of hardware and I/O channels involved demands applications of computer control to achieve the consistency of experimental setup. Thus, the accelerator control system environment stipulates interdisciplinary research including the methodology of end-user programming in order to handle the problems of large-scale control software development [Mejuev *et al*. 2001].

One way to deal with this problem is to apply the techniques of domain modeling and form-based programming [Mejuev *et al*. 1995; Mejuev *et al*. 1997], however this approach would probably require certain skills of the end-users, including the knowledge of object-oriented programming paradigm and rule-based specifications, which is not always acceptable.

On the other hand accumulated experience with software maintenance for experimental physics shows that the most frequent modification requests are targeting relatively small GUI of application logic updates. These modifications could be done by end-users themselves if the appropriate tools are provided.

In this section it is proposed to introduce the notion of tailoring interface in the process of developing software in the accelerator control system environment. A feasibility evaluation was performed with Web-based monitoring applications for 12 GeV Proton Synchrotron at KEK (High Energy Accelerator Research Organization). Using this example the issues of design and implementation of tailorable applications for accelerator control are considered.

The application of technology of end-user tailoring can significantly reduce the time required to perform software modifications during control hardware and software upgrades and correspondingly decrease the overall system maintenance costs, which are high for large installations in the experimental physics.

4.1.2 Requirements

Online monitoring application for KEK Proton Synchrotron should provide display for the beam parameters, accessible in the Java applet environment. The requirements on the design of monitoring application are summarized as follows:

1. The applet should provide a high degree of cross-platform portability, thus it should not rely on native libraries or OS-specific APIs

2. The monitoring system must provide integration with third-party commercial software: Wonderware InTouch, which is widely deployed at KEK Proton Synchrotron.

3. The required *degree of tailorability* for monitoring application is identified as the possibility for the end-users to dynamically reassign mappings of GUI components to I/O channels and customize visual preferences (color, layout) for the components. Some I/O channels are required to display permanently, so that the tailoring functionality should be disabled for the corresponding GUI objects.

The layout of the remote monitoring system is represented in Figure 4.1. The front-end of data acquisition system is represented by PLCs (Programmable Logic Controller). PLC data are accessible through the commercial software – InTouch I/O server that allows data retrieval via NetDDE connections for Windows clients. The Scout Outpost implements a CGI interface accessible from multiplatform Web-based clients. Having received a CGI GET query, the Outpost retrieves the data from I/O server via NetDDE and replies the results in the form of HTML table, including "tag" names, error codes and current values for the "tags", identifying the I/O channels.

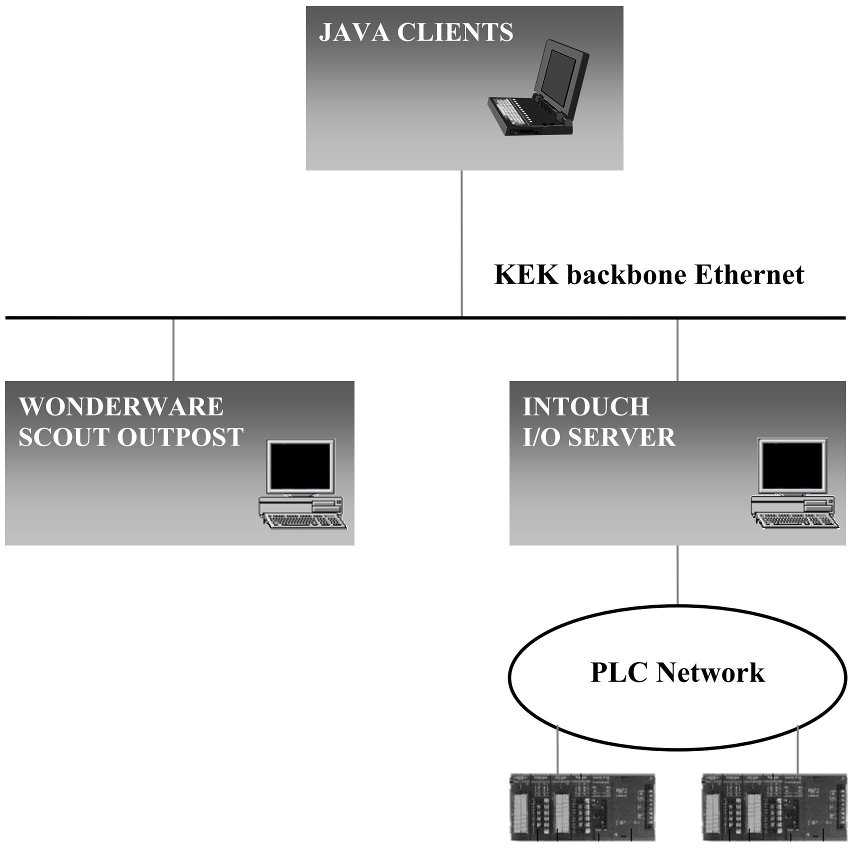
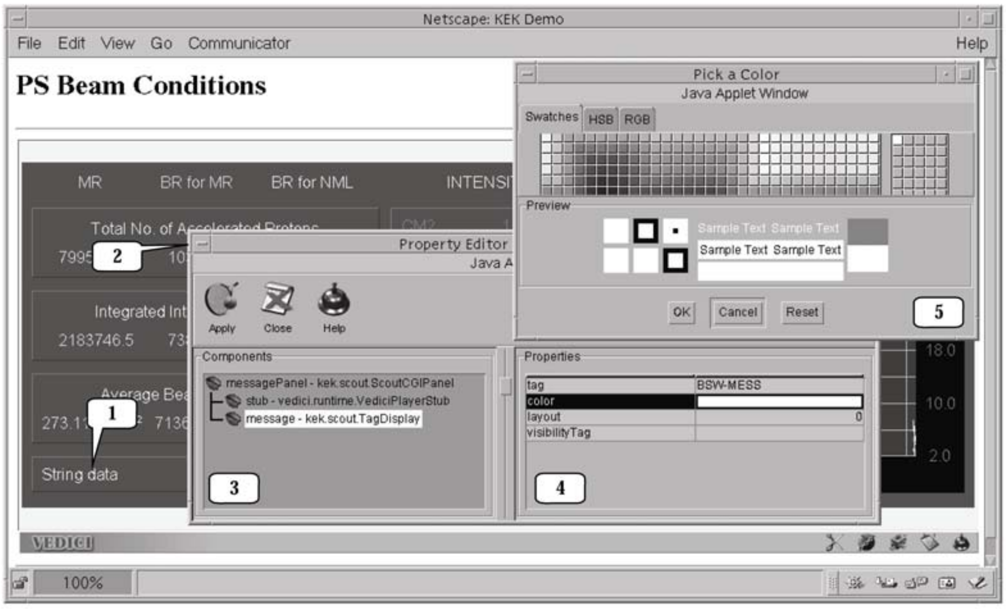


Figure 4.1: Outline of Remote Monitoring System

4.1.3 Implementation

The implementation of dynamic monitoring components was reused from earlier development made with Java 1.0 for the JLC X-band High Field Experiment [Mejuev *et al.* 1998; Higo *et al.* 1998]. The reuse required the conversion of Java 1.0 class libraries into Java Beans, conforming to updated APIs in Java 1.3. The data update is performed by a dedicated component, which wraps Scout Outpost interface and provides a refresh manager for dynamic monitoring components with approximately 1Hz refresh rate. The update manager performs data polling by sending batch requests to the Scout Outpost server.

The monitoring application allows customization of the visual preferences for the UI widgets (color, layout) at runtime and mappings of I/O channels to the widgets. The monitoring application enables each user to define his or her personal view, which includes only the beam parameters of interest. Application developers can optionally disable tailoring functionality for particular groups of widgets of the monitoring application by assigning "visualizer stubs" to the corresponding application fragments.



messagePanel - kek.scout.ScoutCGIPanel
stub - vedici.runtime.VediciPlayerStub
message - kek.scout.TagDisplay

tag	BSW-MESS
color	
layout	0
visibilityTag	

Figure 4.2: Tailorable Remote Monitoring Application

An example of end-user tailoring is presented in Figure 4.2. The application is organized into a hierarchy of nested players each representing an UI panel that contains logically related groups of widgets. In the figure the user is customizing the color of a UI component I n a panel (1) via the "Property Editor" visualizer (2). This visualizer displays all of the player's components in a tree view (3) and allows customization of each component's properties via a corresponding property sheet (4). The property editor is a generic (reusable) visualizer that supports common data types in Java such as *String*, *Boolean*, numbers and *Color* (5).

4.1.4 Summary

The feasibility study described in this section has been performed with WWW-based monitoring application; however the same technique is applicable to the design of control and data-acquisition applications, by extending the framework with a wider set of components. Current implementation uses polling of CGI server that creates redundant traffic in the laboratory network. In the future version it seems reasonable to consider replacing the polling with server "push" interface, which is based on the existing Java implementation of shared data channels [Mejuev and Abe 1997].

4.2 Distance Learning Application

Interactivity and tailorability are considered to be the features required for adding real value to traditional distance learning [Benyon *et al*. 1997; Laurillard *et al*. 1998]. Typically, students study in different environment and they have a variety of browsers with different facilities. To increase the quality of learning, developers of multimedia courseware for the Internet need to provide students with the possibility to be involved into the educational process, rather than simply browse. Interactivity of courseware can be interpreted as a functionality implementation similar to a lecturer posing questions directly to a student and tailorability – as the activity by which a student is able to receive personalized representation of educational content.

4.2.1 The Problematic

The problems with developing multimedia courseware for the Internet or offline study originate in the insufficiency of existing general-purpose software technologies as well as in the mismatch between the learning scenario supported by software tools and traditionally practiced methodologies of academic training. The main problems can be summarized as follows:

1. Interoperability – the software systems are often developed to support only particular lecture or training course, without paying any attention to the possibilities for future reuse elsewhere.

2. Standardization – existing standards on the learning media [Edutool.com 1999; IMS] are rather generic in an attempt to cover a wide range of systems – that makes it difficult to adopt the standards to the real needs of the educators and learners.

3. "User resistance" issues [Hirschheim and Newman 1988], such as scapegoating, a tendency to blame a computer system for any troubles encountered in the process of exploitation of a distance learning systems ("I could not succeed because this system is not efficient").

4. The tendency to limit the objectives of a software development project to a simple hypertextualisation of the existing teaching materials, neglecting the new possibilities offered by emerging technologies.

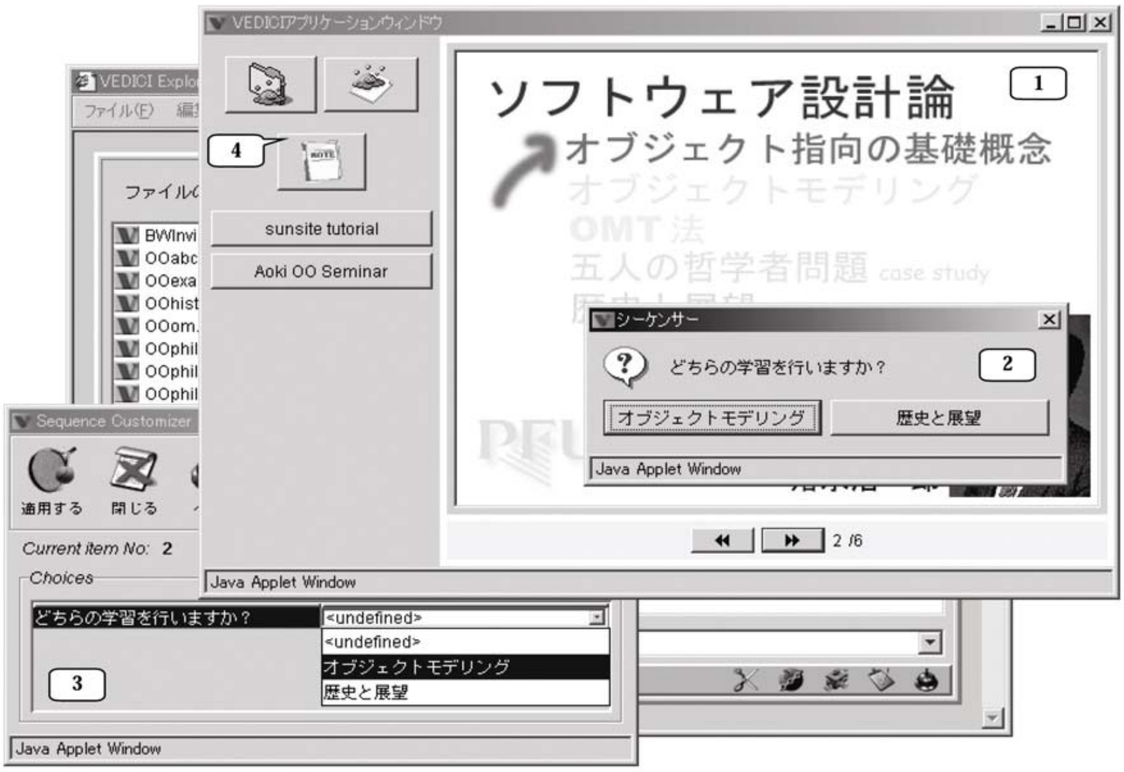
It should be noted that providing guidelines usable for the development of educationally sound multimedia courseware is a complicated issue, involving the problem of developing socio-technical frameworks within a given educational institution [Laurillard *et al.* 1998].

4.2.2 Implementation

A Web-based distance-learning application, developed with VEDICI, covers an introduction to object oriented design concepts and is organized into a hierarchy of nested lectures. Each lecture is defined in a separate CMS and contains an ordered set of slides managed by an instance of the *eLearning.Sequencer* class. The Sequencer can associate an instance of VEDICI player with each slide, so that the player launches its script when the slide is shown. Associations of players with the slides can enhance a lecture with video, simulations and implement the nesting of lectures. Sequencer also defines a set of conditional and unconditional links among the slides that allows dynamically modifying the order in which the slides are displayed. Conditional links specify dialogs that are shown to the user if a corresponding slide is activated. At runtime, the user's responses (choices) can be memorized in the internal state of the Sequencer and later used to activate other conditional links without repeatedly putting the same question to the user. The policy for memorizing users' choices is specified as an attribute of a condition - it can be either "permanent" (user asked once) or "temporary" (user asked always). Definitions of links, conditions and available choices are specified in the CMS of a given lecture. The internal state of a Sequencer is manipulated through getters and setters of its "state" property, the type of this property is a string of proprietary format inherited from a legacy system.

An example of user interaction with the application is presented in Figure 4.3. A dialog defined by a conditional link associated with a slide (1) presents a question "Where do you want to go next?" providing a set of buttons with chapter titles (2) to choose from. The problem with this approach to implementing interactivity is that there is no way to undo or modify the state of a permanent condition once it has been chosen. Moreover, it does not allow visualizing the state of all conditions in a given lecture. These problems are addressed in the following way - at runtime, the user can initiate tailoring for a given lecture applying a visualizer called "Sequence Customizer" (3) to the CMS of the lecture. The Sequence Customizer analyses the CMS in order to locate definitions of the conditions and choices available for each condition. This visualizer then retrieves current condition states by parsing the "state" property of Sequencer and provides a UI for modifying the states of the conditions. The proprietary format of the Sequencer "state" property requires the implementation of a customized persistence scheme for this application - in addition to

CMSs, VEDICI Repository stores a "state file" per each Sequencer. The Repository is also used in this application to store users' personal notes (4).



どちらの学習を行いますか？

Current item No: 2

どちらの学習を行いますか？ <undefined>

Figure 4.3: Distance Learning Application

4.2.3 Summary

As it is commonly acknowledged, in order to improve the effectiveness of a distance learning application, the entire system must be considered as a socio-technical framework. In this context, the practical application of end-user tailoring technology toward the delivery of pedagogically sound courseware requires the building of a conceptual knowledge base covering the pedagogical aspects of distant education. This knowledge base should include interaction models simulating those that occur in traditional lectures and seminars or an investigation of new forms, specific for distance learning. In the case of end-user tailorable applications the expected result of this consideration is the ability to identify the required degree of tailorability in the distance learning domain.

The challenging part of efficiency consideration, in this sense, is that evolving trends of distance learning related technology appears to be causing a shift in the structure of academic teaching from that of a supervisor with personal responsibility for a group of students to that of a knowledge facilitator tasked with providing expert input to a software production team [Laurillard *et al*. 1998].

Regarding the social aspects of distance learning, the application of runtime tailoring and personalization technology can help to reduce or eliminate the extremely high dropout rates typical of early Web-based distance learning programs [Wright and Lee 1999].

Chapter 5

Related Work

This chapter identifies alternatives to the approach presented in this work and compares VEDICI to systems of related scope. The systems described here are significantly different in design, required implementation efforts and areas of applications, which complicate a direct comparison. Additionally, there is no commonly accepted quantitative metric applicable for measuring the efficiency of tailoring interfaces and for verification of necessity of tailoring functionality in a particular application domain in general. Correspondingly, the systems are compared based on their capabilities to support smooth evolution of tailoring interfaces through the application lifecycle, integrate multiple tailoring interfaces within an application instance and availability for reuse in the application domains different from the originally intended ones.

5.1 Alternatives to Component-Based Tailoring

The first alternative that should be pointed out is the use of a hardcoded implementation of tailoring functionality for delivering a particular application instead of building a generic tailoring framework. The "ad hoc" implementation may be preferable if the development is not targeting a sufficiently wide class of systems and the chances are high that appropriate tailoring interfaces will be delivered in the first working version of the application. Contrary to the above, delivering tailorable applications as instances running in a more or less generic compositional framework avoids the duplication of development efforts in the long-term perspective and allows refining some components that may be reused by applications and, potentially, across multiple application domains.

Once the need for a generic compositional framework is established, there is an issue of how the compositional tailoring is implemented in the framework. An alternative to the component-based composition used by VEDICI is feature composition [Teege 2000]. From the users' point of view, a feature is added to the system by simply specifying its presence. This approach reduces the complexity of tailoring; however it discards the ability to specify the relations between components. Tailoring of UIs is considered as an example where feature composition is not directly applicable. A hybrid approach considered as a remedy [Koch and Teege 1999] makes features applicable to several or all existing components.

VEDICI does not sup port feature composition directly, but it can be emulated by application-specific visualizers, although they require significant implementation efforts. For instance, considering the distance-learning application described above, we could define a "video" feature as an availability of video associated with a lecture. Users accessing the application via a slow connection would want to turn the video off globally or, for individual slides. A custom visualizer with this facility should traverse the hierarchy of nested compositional markup specifications, identifying those responsible for video and applying the corresponding changes.

5.2 Component-Based Frameworks

While comparing VEDICI to component-based frameworks that support end-user tailoring, three typical representatives of existing classes of these systems have been considered: a tailorable tool for cooperative work implemented as a "native" application (OVAL), a Web-based tailorable system with HTML forms interface (ICE) and a Java framework for component based tailorability of Computer Supported Cooperative Work (CSCW) applications (EVOLVE).

5.2.1 OVAL

OVAL [Malone *et al.* 1992] is a radically tailorable tool for cooperative work that was implemented with Macintosh Common LISP for the Macintosh operating platform. OVAL applications are composed from *objects*, *views*, *agents* and *links*. The appearance and functionality of OVAL is similar to that of Lotus Notes, where documents are represented as semi-structured templates with user-definable views. However, in comparison with Lotus Notes, OVAL is more generic and customizations in OVAL can be performed by end-users, with minimal skills required.

OVAL focuses on providing end-users with the possibility to re-design working applications rather than on supporting runtime tailoring. Case studies introduced in [Malone *et al.* 1992] have been carried out with groupware systems and required modifications in the underlying OVAL framework to fully accommodate each system. Contrary to this, the applications of VEDICI described in this thesis are running within an invariant framework which interfaces with application-dependent code (visualizers, repositories) via a set of fixed APIs and provides abstract classes and documentation to simplify application programming.

5.2.2 ICE

ICE [Farshchian 1998], is a Web-based system in support of collaborative environments on the Web. ICE is accessible via Web browsers, so no installation is required on the client side. Using ICE, the Internet users can compose groupware applications from information, collaboration and interface objects.

ICE uses an HTML-based user interface and in this sense VEDICI Runtime can provide a better quality UI since it uses JavaBeans. VEDICI facilitates integration with Web browsers and can be used to emulate some of the functionality of ICE. In addition, the building blocks and tailoring interface in ICE are predefined, thus restricting the reusability of the entire system.

5.2.3 EVOLVE

EVOLVE [Stiemerling *et al*. 1999] was developed in support of distributed CSCW applications. In order to avoid runtime code generation, a component model in EVOLVE has been redefined from JavaBeans to a platform-specific FLEXIBEANS model. The composition of FLEXIBEANS in EVOLVE is specified by a proprietary configuration language (CAT files).

It has been reported [Stiemerling *et al*. 1999] that EVOLVE employs visual programming techniques to be used for end-user tailoring and that the usability of the platform has been studied in a users' workshop environment. The workshop identified the need to experiment with different styles of visual programming in order to identify the ones that are acceptable to end-users. However, this need is not directly addressed by the architecture of EVOLVE; in addition, tailorable applications in the real-world often require a proprietary tailoring interface to be integrated into a framework. A simple example of this is a text editor with customizable toolbar. In this case, a well-known metaphor of moving the buttons between two lists would be preferable to the contrary fully-fledged visual programming technique.

5.3 Summary

Advantages of the approach proposed in this work can be summarized as follows:

1. In VEDICI the tailoring framework is defined as an external entity, which can reflect on the structure of a compositional application and invoke the tailoring interfaces (defined independently from the framework) to modify the composition and attributes of primitive components. This decoupling of tailoring interfaces and runtime components can be considered as a key point in achieving the reusability of proposed framework across the boundaries of application domains.

2. The applications are partitioned into a hierarchy of composite components, representing entry points available for applying multiple tailoring techniques, rather than strict binding between the components' structure and tailoring interfaces.

3. Common services such as persistence, nesting and integration with the execution environment (Web browser) are implemented in an application independent fashion, simplifying the migration of framework to other domains.

4. Framework implementation is based on standard technologies and is easy to deploy and extend.

The case studies, performed in this work showed the feasibility of implementing tailorable applications based on VEDICI in the domains of distance learning and remote monitoring in experimental physics. The fact that the same framework could be applied for making working systems in these completely different fields proves the reusability and generic nature of the proposed framework.

Chapter 6

Conclusions and Perspectives

This thesis examined a process of developing tailorable Web applications and identified the problems relating to this development. The use of compositional markup specifications and the integration of multiple tailoring interfaces have been proposed as measures for dealing with problems of the development. The thesis introduced the composition and modules of a generic reusable platform for tailoring and described the application of this platform in the development of multimedia courseware for the Internet and remote monitoring applications in accelerator physics.

VEDICI currently employs a customized component-scripting language to instantiate primitive application components and to define bindings among components. However, the problem of preserving the *intrinsic states* of objects is not inherently addressed within compositional markup specification languages, which replace recording of object internal states with reconstituting the states with public APIs. The problem with intrinsic states can become severe when dealing with legacy component libraries, or implementing bean wrappers for legacy Java code. This problem can be solved with additional implementation efforts by introducing a "state" property and, implementing export and import of intrinsic state in its getter and setter methods. Another possible solution is to open the component "black box" using meta-protocol techniques [Kiczales 1996], a solution that would require the introduction of a proprietary component model for the framework. Deviation from JavaBeans compatibility would in turn require the implementation of wrappers for each component that can be used within the framework, including Java AWT and Swing UI components.

The biggest obstacle to a tailorable software system being adopted by a given organization, however, does not relate to the problems of software technology. It is the difficulty of integrating the system into its social context and system compatibility with already established behavioral patterns. A tailorable system has an additional potential to solve the issues related to system adaptation and evolution, but it only works if the end-users are sufficiently motivated to carry out tailoring in the first place. An explicit request originating from the end-users for tailoring facilities allows the assumption that such a motivation exists a priori. In addition to a well-known CSCW, this thesis represents a software environment of a large-scale experiment in accelerator physics where tailoring is explicitly required and planned for. The development of Problem Solving Environments (PSE) for scientific computation and engineering can be seen as another potential domain where scripting "Webware" and interactive interface development technologies are demanded [Rice and Boisvert 1996].

Following sections introduce future perspectives for applications of the proposed framework to the development of adaptive tailorable systems and for carrying out implicit usability inspections.

6.1 Adaptive Tailorable Frameworks

Application of tailoring frameworks in general (and VEDICI in particular) to the development of customizable applications for the Web creates a set of options to support the evolution of an application after deployment. This evolution is certainly driven by the needs of end-users; however the developers of the system retain full control on the process of modifications. The key factor motivating a change in the composition of a deployed application is the intent to meet the requirements of a larger community of users, minimizing the differences between personalized applications and initial templates (both are stored in an application repository in the case of VEDICI).

Thus, the deployment of a tailorable application can not be clearly separated from the development, it rather states for field evaluation. The incremental phases of development/deployment software process allow reusing tailoring framework for the tasks ranging from relatively small ergonomic UI updates to the most extreme incremental approach, defining the increments from the top of the software life cycle.

The representation of such a deployment process in terms of evolutionary software process model, described in the Chapter 2 identifies the following phases:

1. Introducing a *functional model* that does not have to consider deeply the HCI aspects of the future system.

2. Deployment of the prototype on the Web, where it becomes accessible for the target audience.

3. Development and clarification of the *user interaction model* through the usability evaluations, which rely on the analysis of tailoring activity of the end-users.

The procedure described above can be potentially applied to the design of *adaptive* tailorable systems, where the changes are performed (semi)automatically, requiring a minimum intervention from professional developers. It would require however a well understood and fixed functional model. Figure 6.1 represents a revised evolutionary process model, which integrates the development and deployment processes, feasible within a generic tailoring framework.

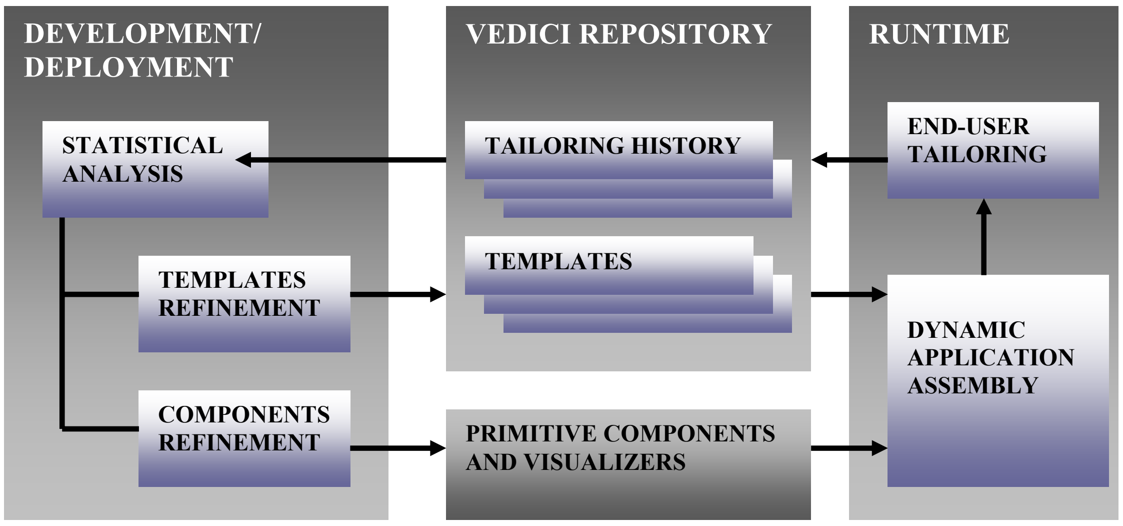


Figure 6.1: Software Process for an Adaptive Tailoring Framework.

6.2 Implicit Usability Inspections

Composition of applications from monolithic components by means of scripting allows a certain degree of flexibility. In a component-scripting application an equal functionality can be achieved in several ways, for example by changing the degree of granularity (producing larger scripts) or by extending component interfaces, so that some functionality previously embedded into script can be accessed by a call to a component's method.

However the scripting applications with equal functionality in the sense described above does not provide at all times equal efficiency (implement an optimal way to achieve a given goal) and usability (provide all required support for the users). To verify that the current decomposition is appropriate, the developers can rely on the analysis of the content of VEDICI repository, which contains all the necessary information for performing of implicit usability inspections.

Usability inspections [Nielsen 1994] represent an essential part of software lifecycle. The discovery of a mismatch between the capabilities of currently deployed software system and ergonomic or functional requirements of the end-users (user interaction model) is a driving force for a software change. However the explicit requests from the end-users cannot be collected in all cases, especially for an application deployed on the Web.

VEDICI provides the possibility to collect the users' feedback implicitly – through the analyzing an individual modification history and statistics over the data available in the server side repository. Data in the repository represent a snapshot revealing the severity of usability problems and the way the problem is being addressed by the end-users. From this point of view tailoring activity can be considered as an indication of a minor or major design mismatch.

Moreover, VEDICI allows collecting a structured feedback, in the form of XML-based application source scripts, modified as a result of tailoring activity. The modification histories can also be applied in usability engineering as a supplement for traditional methods of collecting users' feedback, such as questionnaires, video recording or eye-tracking.

Bibliography

[Appelt *et al*. 1998] Appelt, W., Hinrichs, E., Woetzel, G., Effectiveness and efficiency: the need for tailorable user interfaces on the Web, in: *Proceedings of Seventh International World Wide Web Conference,* Brisbane, Australia, 1998.

[Benyon *et al*. 1997] Benyon, D., Stone, D., Woodroffe, M., Experience with developing multimedia courseware for the World Wide Web: the need for better tools and clear pedagogy, *International Journal of Human-Computer Studies*, 47 (1), Academic Press Inc., 1997, pp. 197-218.

[Cypher 1993] Cypher, A., ed., *Watch What I Do: Programming by Demonstration*, MIT Press: Cambridge MA, 1993.

[Edutool.com 1999] Edutool.com, Learning Technology Systems Architecture (LTSA); http://edutool.com/ltsa/ (1999).

[Farshchian 1998] Farshchian, BA., ICE: An object-oriented toolkit for building collaborative Web applications, *Proceedings of the IFIP TC8/WG8.1 Working Conference on Information Systems in the WWW Environment*, Chapman & Hall: Beijing, 1998, pp. 70-86.

[Floyd *et al*. 1989] Floyd, C., Reisen, F.-M., and Schmidt, G., STEPS to Software Development with Users, in: *Lecture Notes in Computer Science*, Vol 387: ESEC '89, ed. C. Ghezzi and J. A. McDermid, Springer-Verlag, 1989, pp. 48-64.

[Gamma *et al*. 1995] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, 1995.

[Higo *et al*. 1998] Higo, T., Dong, D., Fang, H., Nie, J., Gao, M., Kadokura, E., Mejuev, I., Sakai, H., and Takata, K., High Field Experiment of 1.3m-Long X-Band Structure; in: *Proceedings of the First Asian Particle Accelerator Conference*, March 23-27, 1998, KEK, Tsukuba, Japan, pp. 169-171.

[Hirschheim and Newman 1988] Hirschheim, R., Newman, M., Information Systems and User Resistance: Theory and Practice, *The Computer Journal*, 31 (5), 1988, pp. 398-408.

[IMS] IMS Global Learning Consortium Inc.; http://www.imsproject.org/

[Johnson 1999] Johnson, M., Bean Markup Language, *JavaWorld*, August 1999.

[Keen 1991] Peter G. W. Keen, *Shaping the Future: Business Design Through Information Technology*, Boston: Harvard Business School Press, 1991.

[Kiczales 1996] Kiczales, G., Beyond the Black Box: Open Implementation, *IEEE Software*, 1996; 13 (1), pp. 8-11.

[Koch and Teege 1999] Koch, M., Teege, G., Support for Tailoring CSCW Systems: Adaptation by Composition, in: *Proceedings of 7th Euromicro Workshop on Parallel and Distributed Processing*, IEEE Press: Funchal, Portugal, 1999, pp. 146-152.

[Laurillard *et al.* 1998] Laurillard, D., Preece, J., Shneiderman, B., Neal, L., Wærn, Y., Distance Learning: Is it the End of Education as Most of Us Know It? in: *Proceedings of the CHI 98 summary conference on CHI 98 summary: human factors in computing systems*, ACM Press: New York, 1998, pp. 86-87.

[MacLean *et al.* 1990] MacLean, A., Carter, C., Lövstrand, L., Moran, T., User-tailorable systems: Pressing the issues with buttons, Chew, J. C., Whiteside, J., (eds.), in: *Proceedings of CHI '90*, ACM Press: New York NY, 1990, pp. 175-182.

[Malone *et al.* 1992] Malone, Th., Fry, Ch., Lai, K.-Y., Experiments with OVAL: A Radically Tailorable Tool for Cooperative Work, in: *Proceedings of the Conference on Computer-Supported Cooperative Work*, ACM Press: New York, 1992, pp. 289-297.

[Megginson and Brownell 2002] D. Megginson, D., Brownell, D., Simple API for XML, http://www.saxproject.org/ (2002).

[Milne 1999] Milne, P., Long Term Persistence for JavaBeans Technology, *Sun's 1999 Worldwide Java Developer Conference (JavaOne)*, June 15-18, 1999, San Francisco, USA.

[Mejuev *et al.* 1995] Mejuev, I., Abe, I., and Nakahara K., Object-Oriented Control System Development Using Smalltalk Language; in: *Proceedings of The 1995 International Conference on Accelerator and Large Experimental Physics Control Systems*; October 25 - November 3, 1995, Chicago, USA, p. 713.

[Mejuev and Abe 1997] Mejuev, I., Abe, I., Java Application for Creating a Shared Object Cash, in: *Proceedings of The 1997 International Conference on Accelerator and Large Experimental Physics Control Systems*; November 3-7, 1997, Beijing, China.

[Mejuev *et al*. 1997] Mejuev, I., Abe, I., Nakahara, K., Application of Smalltalk Language for Accelerator Control; *Nuclear Instruments & Methods in Physics Research*, A 389 (1997), pp. 38-41.

[Mejuev *et al*. 1998] Mejuev, I., Kumagai, A., Takahashi, M., Kadokura, E., Higo, T., and Takata, K., Status of Control and Data Acquisition System for JLC X-Band High Field Experiment; in: *Proceedings of The 23rd Linear Accelerator Meeting in Japan*; September 16-18, 1998, Tsukuba, Japan, p. 367.

[Mejuev *et al*. 2000] Mejuev, I., Higashida, M., Shimanaka T., Makino, N., Integration of Multiple Tailoring Interfaces in Compositional Web Applications, in: *Proceedings of 2000 International Conference on Information Society in the 21st Century: Emerging Technologies and New Challenges*, November 5-8, 2000, The University of Aizu, Japan.

[Mejuev *et al*. 2001] Mejuev, I., Higashida, M., Shimanaka, T., Makino, N., Integration of Multiple Tailoring Interfaces in Compositional Web-Based Applications, in Enabling Society with Information Technology, Jin., Q., Li, J., Zhang, N., Cheng, J., Yu, C., Noguchi, S. (eds.), Springer-Verlag: Tokyo, 2001, pp. 111-121.

[Mejuev *et al*. 2001] Mejuev, I., Kumagai, A., Kadokura, I., Tailorable Software Architectures in the Accelerator Control System Environment, in: *VII International Workshop on Advanced Computing and Analysis Techniques in Physics Research, AIP Conference Proceedings*, vol. 583: ACAT'2000, Bhat, P., Kasemann, M., (eds.), Melville: New York, 2001, pp. 119-121.

[Mørch 1997] Mørch, A. Three Levels of End-User Tailoring: Customization, Integration, and Extension; in: *Computers and Design in Context*, eds. Kyng M, Mathiassen L.; The MIT Press: Cambridge, MA, 1997; pp. 51-76.

[Mørch *et al.* 1998] Mørch, A., Stiemerlieng, O., Wulf, V., Tailorable Groupware, *ACM SIGCHI Bulletin*, Vol. 30, No. 2, April 1998.

[Nielsen 1994] Nielsen, J., Usability Inspection Methods, in: *Proceedings of CHI '94 Conference Companion on Human factors in computing systems*, 1994, pp. 413-414.

[Nierstrasz *et al*. 1991] Nierstrasz, O., Tsichritzis, D., de Mey, V., Stadelmann, M., Objects+Scripts=Applications, in: *Proceedings of Esprit 1991 Conference*, Kluwer Academic Publishers, 1991, pp. 534-552.

[Rice and Boisvert 1996] Rice J., Boisvert R., From Scientific Software Libraries to Problem Solving Environments, *IEEE Computational Science and Engineering*, 3 (3), 1996; pp. 44-53.

[Shimanaka *et al*. 2000] Shimanaka, T., Mejuev, I., Higashida, M., Makino, N., VEDICI: A Framework for Developing Distributed Component-Based Applications, in: *Proceedings of SEA Software Symposium'2000*, June 21-23, 2000, Kanazawa, Japan, pp. 16-19 (in Japanese).

[Sommerville 1996] Sommerville, I., Software Process Models, *ACM Computing Surveys*, Vol. 28, No. 1, March 1996, pp. 269-271.

[Stallman 1981] Stallman, R., EMACS, the extensible, customizable, self-documenting display editor, in: *Proceedings of ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland OR, 1981.

[Stiemerling *et al*. 1997] Stiemerling, O., Kahler, H., Wulf, V., How to Make Software Softer - Designing Tailorable Applications, in: *Proceedings of DIS'97*, Amsterdam, August 18 - 20, 1997, pp. 365-376.

[Stiemerling *et al*. 1999] Stiemerling, O., Hinken, R., Cremers, Armin B., Distributed Component-Based Tailorability for CSCW Applications, in: *Proceedings of the ISADS '99*, IEEE Press: Tokyo, Mar. 20-23, 1999, pp. 345-352.

[Sun 2002] http://java.sun.com/j2se/1.4/docs/api/java/beans/XMLDecoder.html (2002).

[Teege 2000] Teege, G., Users as Composers: Parts and Features as a Basis for Tailorability in CSCW Systems, *Computer Supported Cooperative Work (CSCW)*, 2000 (9), pp. 101-122.

[W3C 1998] World Wide Web Consortium (W3C), Document Object Model (DOM), *W3C Recommendation*, http://www.w3.org/DOM/ (1998-2003).

[W3C 2000] World Wide Web Consortium (W3C), Extensible Markup Language (XML) 1.0 (Second Edition), *W3C Recommendation*, http://www.w3.org/TR/REC-xml (2000).

[Williams 1987] Williams, G., HyperCard: HyperCard extends the Macintosh user interface and makes everybody a programmer, *Byte*, 12: 109-117, Dec. 1987.

[Wright and Lee 1999] Wright, S. W. Y., Eleonor Lee, Distance Learning, *Community College Week*, 11(22), 1999, pp. 6-9.

Publications

A Java-based EPICS Archive Viewer with SOAP Interface for Data Retrieval; Furukawa, K., Sato, M., Mejuev, I.; *The 2003 International Conference on Accelerator and Large Experimental Physics Control Systems*, Gyeongju, Korea (under review).

Developing end-user tailorable Web applications using a compositional framework; Mejuev, I.; *Software – Practice and Experience*, Wiley (in press).

Integration of Multiple Tailoring Interfaces in Compositional Web Applications; Mejuev, I., Higashida, M., Shimanaka, T., Makino, N.; *Proceedings of The 2000 International Conference on Information Society in the 21st Century: Emerging Technologies and New Challenges*, November 5-8, 2000, The University of Aizu, Japan, pp. 169-175
Also published in a book: *Enabling Society with Information Technology*, eds., Q. Jin, J. Li, N. Zhang, J. Cheng, C. Yu and S. Noguchi, Springer-Verlag, Tokyo, 2001, pp.111-121.

Tailorable Software Architectures in the Accelerator Control System Environment; Mejuev, I., Kumagai, A., Kadokura, E.; VII International Workshop on Advanced Computing and Analysis Techniques in Physics Research, *AIP Conference Proceedings*, vol.583: ACAT'2000, eds. P. Bhat and M. Kasemann, Melville, New York, 2001, pp.119-121.

Application of Smalltalk Language for Accelerator Control; Mejuev, I., Abe, I., Nakahara, K.; *Nuclear Instruments & Methods in Physics Research*, Section A, 389(1-2), Elsevier, 1997, pp. 38-41.

Modeling the Behavior of Complex Systems Using Statistical Regularities; Mejuev, I., Dudikhin, V.; *NTI, Ser. 2, Informational Processes and Systems*, №10 (1994), pp. 27-29. (in Russian)