MyUI: Mainstreaming Accessibility through Synergistic User Modelling and Adaptability

FP7-ICT-2009-4-248606

User Interface Adaptation Engine Prototype

Public Document

<table>
<thead>
<tr>
<th>Deliverable number</th>
<th>Date of delivery</th>
<th>Status</th>
<th>Type</th>
<th>Workpackage</th>
<th>Authors</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2.3</td>
<td>02-2013</td>
<td>Final</td>
<td>Prototype</td>
<td>WP 2 - Self-adaptive Multimodal User Interfaces</td>
<td>FHG (Matthias Peissner, Daniel Ziegler, Dagmar Häbe, Andreas Schuller)</td>
<td>Adaptation engine; adaptive user interface; user interface generation</td>
</tr>
</tbody>
</table>

Abstract

This document summarizes the implementation of the MyUI Adaptation engine and provides an illustration of its functioning based on screen shots from example use cases. The MyUI Adaptation Engine generates personalised user interfaces for optimized accessibility and adapts the user interface during run time. The adaptation engine covers longer-term individual user requirements and short-term variations in the current context of use by generating and adapting personalized user interfaces during the interaction according to an evolving user and context profile.

© 2010-2013 MyUI Consortium
Table of Contents

EXECUTIVE SUMMARY.............................................................................................................. 4

1. INTRODUCTION..................................................................................................................... 5
   1.1 THIS DOCUMENT.................................................................................................................. 5
   1.2 THE TASK IN RELATION TO THE MyUI Project............................................................... 5

2. INTRODUCTION TO THE MyUI ADAPTATION ENGINE ............................................ 6
   2.1 EXTENSIVE ADAPTATIONS IN A Model-View-Controller ARCHITECTURE ............... 6
   2.2 MODULARITY AND CONSISTENT USER INTERFACES .................................................... 7
   2.3 ADAPTATIONS DURING USE ON THE BASIS OF SELF-LEARNING USER AND CONTEXT
       PROFILES ............................................................................................................................. 8
   2.4 TRANSPARENCY AND CONTROLLABILITY FOR THE END USERS................................. 8

3. MYUI ADAPTATION ENGINE IN THE MYUI INFRASTRUCTURE ............................ 8
   3.1 SYSTEM ARCHITECTURE AND COMPONENTS ............................................................... 8
   3.2 PLUGIN APPROACH OF THE MYUI INFRASTRUCTURE .................................................. 10

4. MYUI ADAPTATION ENGINE IN THE THREE-STAGE PROCESS OF UI
   GENERATION AND ADAPTATION ......................................................................................... 11
   4.1 USER INTERFACE PARAMETERIZATION ........................................................................ 12
      4.1.1 Summary of the UI Parameterization concept ............................................................. 12
      4.1.2 Implementation: profiles_controller......................................................................... 13
   4.2 USER INTERFACE PREPARATION ................................................................................... 15
      4.2.1 Summary of the UI Preparation concept ................................................................... 15
      4.2.2 Implementation: prepareAdaptation ........................................................................ 16
   4.3 USER INTERFACE GENERATION AND ADAPTATION .................................................... 17
      4.3.1 Summary of the UI Generation and Adaptation concept ........................................... 17
      4.3.2 Implementation: performAdaptation ....................................................................... 18

5. IMPLEMENTATION OF BASIC MYUI CONCEPTS......................................................... 19
   5.1 INTERACTION SITUATIONS.............................................................................................. 19
   5.2 INTERACTION PATTERNS AND USER INTERFACE ELEMENTS .................................... 19
   5.3 INTERACTION PATTERNS AND ADAPTATION HANDLERS .......................................... 21
   5.4 ADAPTATION PATTERNS AND UNDO ADAPTATIONS ................................................... 22
      5.4.1 Context-specific selection of the appropriate adaptation pattern ................................ 22
      5.4.2 Help the end user to anticipate the effects of adaptations ....................................... 23
      5.4.3 Undo the latest adaptation ....................................................................................... 24
   5.5 MAIN FUNCTIONS OF THE ADAPTATION PROCESS .................................................... 24

6. MYUI ADAPTATION ENGINE IN ACTION – EXAMPLE USE CASES ....................... 26
   6.1 MAIN MENU ................................................................................................................... 26
   6.2 WEATHER APPLICATION ............................................................................................... 28
6.3 USER ACCESS TO THE USER INTERFACE PROFILE ................................................................. 29
6.4 SETTING UP THE MYUI RUNTIME ENVIRONMENT ................................................................. 30
   6.4.1 MyUI Core and website .............................................................................................. 30
   6.4.2 Context Manager Infrastructure ................................................................................. 30
7. CONCLUSION ......................................................................................................................... 31
REFERENCES ............................................................................................................................... 32
Executive Summary

This deliverable is the documentation of the MyUI prototype D2.3 “User Interface Adaptation Engine Prototype”. It describes the implementation of the adaptation engine which reflects the innovative concept for adaptive MyUI user interfaces (D2.2). The adaptation engine is the core element of the MyUI framework and infrastructure for adaptive user interfaces which aim at increasing the usability and accessibility of ICT products for people with special needs by providing dynamic and highly individualized user interfaces.

Based on a summary of the MyUI adaptation concept and respective requirements of the adaptation engine (section 2), the adaptation engine is described from different perspectives:

- the adaptation engine within the MyUI system architecture and the used CakePHP plugin approach (section 3),
- major functions of the adaptation engine in the three-stage process of user interface generation and adaptation (section 4), and
- the implementation of core MyUI concepts such as interaction situations, adaptation handlers, different design patterns including adaptation patterns and main functions of the adaptation process (section 5).

Finally, this documentation includes sample screenshots of generated user interfaces to illustrate the results from the end users’ point of view. Section 6 illustrates how different changes in the MyUI user profile (UI) lead to different variants of the user interface for the same interaction situation. The included screenshots provide only a small impression of the diversity of possible user interface adaptations with the MyUI Adaptation Engine.
1. Introduction

1.1 This Document

This is the public documentation of prototype D2.3 “User Interface Adaptation Engine Prototype” which is a major deliverable of the MyUI project (ICT-248606).

The results are specifically the outcome of Task 2.4 “User Interface Adaptation Engine” which builds on Task 2.2 “Concepts for self-adaptive multimodal user interfaces”.

1.2 The task in relation to the MyUI project

The conceptualisation and implementation of adaptive user interfaces for users with special needs are major tasks in the MyUI project. The effectiveness and quality of the adaptive user interface solutions are essential for the success of the entire project. Therefore, the prototype D2.3 is a core deliverable of the project.

Important input to this task and prototype stems from the following project activities and documents:

Task 2.2 and Task 2.3 with Deliverable D2.2 “Adaptation concept and Multimodal User Interface Patterns Repository”:
- Three stage process of UI generation and adaptation
- Design Patterns and their roles in the adaptation process
- Abstract Application Interaction Model (AAIM) with its Interaction Situations as an anchor for the variants of the adaptive user interface
- Adaptation patterns as a means to transparency and controllability for the end user
- The MyUI design patterns repository as a specification of the reusable user interface components

Tasks 1.3 with Deliverable D1.2 “Prototype for user context management infrastructure (UCMI) and user modelling”:
- Providing up-to-date user profiles as a basis for the adaptation engine’s work
- Bi-directional interfaces between adaptation engine and UCMI

Adaptive systems can be considered to consist of three main levels (Weibelzahl, 2002; summarising Oppermann, 1994 and Jameson, 2001):
- Afference – collection of observational data about the user.
- Inference – creating or updating a user profile based on that data.
- Efference – deciding how to adapt the system behaviour.

The work in WP 2 concentrates on the efference functions of the MyUI adaptive system. Afference and inference are covered by WP 1 and described and implemented in Deliverables D1.1 and D1.2. However, the adaption engine supports afference functions by providing information about relevant user interface events. The modules developed in WP 2 and described in this document assume to start from a somehow reliable user profile. The mechanisms which collect and interpret information about the user and his context are not focused in this document but subject of WP 1 (see Figure 1).
The main challenge of the development work summarized in this report is the implementation of a prototype which illustrates the innovative aspects of the MyUI project by generating and adapting user interfaces during run-time which allow end users with special needs to interact with a system (here iTV device with services) in a comfortable manner.

2. Introduction to the MyUI Adaptation Engine

The adaptation engine is the core element of the functional MyUI infrastructure for adaptive user interfaces. It is responsible for generating and adapting user interfaces to cover individual user needs and context requirements. The functionality and requirements of the MyUI adaptation engine reflect the major principles of the MyUI adaptation concept for generating accessible user interfaces (cf. D2.2). These principles can be summarized as follows:

2.1 Extensive adaptations in a Model-View-Controller architecture

A generic framework for accessible user interfaces must provide mechanisms to overcome any kind of barriers of use. These can include

- Diverse user skills, capabilities and needs,
- Diverse environmental conditions of use, and
- Diverse technical setups.

This triple of user, environment and technical platform is often referred to as the “context of use” (cf. Calvary et al., 2011). When considering possible motor, perceptual and cognitive user impairments it gets clear that very different and extensive user interface adaptations will be required to overcome respective barriers. Therefore, MyUI covers user interface adaptations in all of the following areas:

- **Presentation formats**, e.g. font sizes, screen layout, presentation modalities¹,

---

¹ We use the term „modality“ in a sense as defined by Vanderdonckt et al., 2008
– **Navigation**, i.e. the possible user paths to access specific functions or content, and
– **Interaction mechanisms**, which includes the support of diverse input devices and input modalities and different ways of interpreting user input.

In order to cover user interface adaptation to this extent, it is necessary to have strict separation of the user interface from the application logic so that one application state can be presented to the end user and manipulated by the end user in very different ways. MyUI is based on the Model-View-Controller paradigm (MVC, cf. Gamma, Helm, Johnson & Vlissides, 1994). The *model* maintains the basic data. The *view* represents the information output perceivable by the end user. And the *controller* processes the user input and executes specific functions which can lead to a change in the model and to an update in the view. By a Publish/Subscribe-structure (cf. Eugster et al., 2003) the interdependencies between model and view can be further reduced. The timeliness of the view can be assured by immediately notifying the view as soon as the controller has changed the model (translated from Peissner, in preparation).

### 2.2 Modularity and consistent user interfaces

“To cover the great heterogeneity of users, environments and devices, MyUI follows a modular approach to user interface development which relies on multimodal user interface design patterns. The MyUI design patterns contain the knowledge needed to perform the (…) process of user interface generation and adaptation. Individual accessibility is achieved by composing design patterns which provide proven solutions for specific interaction situations and characteristics of the user, environment and device. Adapting the user interface means switching from one design pattern of a bundle (e.g. all patterns for single selection from a list of options) to another pattern of the same bundle which is hypothesized to be the most appropriate for the current context. (…) Each pattern is linked to a reusable software component and associated with a source code representation of the described solution to put the recommended guideline into action in the MyUI adaptive user interface.” (Peissner et al., 2012a).

Besides this concept of modularity, MyUI recognizes the importance of consistency in the user interface. Whereas, some of the design patterns (the Interaction Patterns and the User Interface Elements) can be regarded as building block (user interface widgets) for the generated user interface, other design patterns set global variables which apply to all areas of one user interface to assure common user interface settings across all MyUI applications and interaction situations of a current user. Examples include device-specific settings such as layout grids, screen resolution and available input techniques which are set by the *Device-specific Patterns* and user interface settings which cover general individual user and environmental requirements such as maximum number of elements per screen (depending on user’s attention capabilities), font sizes, etc. as set by the *Individualization Patterns* (Peissner et al., 2012b).

These two principles of modularity and consistency are reflected also by the two major responsibilities of the adaptation engine:

1. First, the MyUI Adaptation Engine coordinates the transformation of the user profile and device profile to the user interface profile. This process step is called UI Parameterization. It is “outsourced” to the MyUI Middleware and performed by the Individualization Patterns and Device-specific Patterns. UI Parameterization is assuring the consistency requirement by managing and manipulating global variables valid throughout the entire user interface.
2. The second responsibility is the generation, rendering and adaptation of the user interface during run-time. This covers the two process steps of UI Preparation and UI Generation and Adaptation. The required design patterns include Interaction Patterns, UI Elements and Adaptation patterns which serve as the building blocks for composition of the modular user interface as directly perceivable by the end-user.
2.3 Adaptations during use on the basis of self-learning user and context profiles

Self-learning user and context profiling and system-initiated adaptations during use are key features of the MyUI infrastructure. Adaptations during use support a direct reaction to newly available knowledge about the user and the environment of use. Moreover, they provide the opportunity to overcome barriers of use immediately when they occur. Changing abilities in the course of aging and rehabilitation can also be covered when providing adaptations during use.

Run-time adaptations and self-learning profiling mechanisms result in the following requirements towards the adaptation engine:

- Permanent connection to the MyUI user and context management infrastructure (UCMI) and the capabilities to recognize changes in this knowledge base.
- Generate a user interface at the beginning of the run time and at each state transition of the application and the capability to adapt the interface during runtime and within one state according to the new context.
- Provide a back channel to the UCMI and the capability to notify the context manager about relevant user interface event, e.g. time-out, undo of recent adaptations, etc.

2.4 Transparency and controllability for the end users

System-initiated runtime adaptations during use can lead to problems for the users. Inconsistencies and changes in the user interface can confuse the user and produce a feeling of losing control which can lead to bad usability and low acceptance (cf. Weld et al., 2003). Therefore, MyUI provides dedicated mechanisms to assure transparency and controllability of automatic adaptations: the MyUI Adaptation Patterns. For a realisation of the adaptation patterns the adaptation engine offers the following capabilities:

- Recognize the need for user interface adaptations and assess the severity of the change, i.e. does the adaptation include switching to a different interaction pattern, and the capability to select and trigger the appropriate adaptation patterns.
- Animations as adaptation rendering mechanisms to smoothly but obviously move from one instance of the user interface to another.
- Offer information about the intended user interface so that the user can explicitly accept or reject an adaptation before the adaptation is carried out.

3. MyUI Adaptation Engine in the MyUI Infrastructure

3.1 System architecture and components

The architecture of the MyUI system comprises the following three sub-systems with different components and responsibilities for the design and implementation of adaptive user interfaces as illustrated in Fehler! Verweisquelle konnte nicht gefunden werden.:
Figure 2 Architecture of the MyUI system

1. **Client**
The client is a specific end user device which provides access to the locally running MyUI services and applications. Input and output devices are connected to the client in order to allow the user to interact with the MyUI system. All devices are registered in the Device Manager of the MyUI Middleware. Hardware and software sensors detect relevant interaction events such as operating errors and timeouts. The processed data is then transmitted to the Sensor Data Manager of the Context Manager. The client includes also the MyUI Adaptation Engine which is responsible for generating, rendering and adapting the user interface. For this purpose the MyUI Adaptation Engine receives updates of the current user interface profile and starts the adaptation process while considering the interaction patterns selected from the MyUI Pattern Repository.

2. **MyUI Middleware**
The MyUI Middleware includes the Device Manager, the Pattern Repository and the UI Parameterization Process. This process provides a user interface profile based on selected Individualization Patterns and Device-Specific Patterns from the Pattern Repository. The user profile received from the Context Manger and the device profile from the Device Manager serve as major inputs for UI Parameterization. The resulting user interface profile is sent back to the client.

3. **Context Manager**
The Context Manager is a locally running software component which maintains the user and environment profile in the User Profile Manager. The user profile includes a set of variables that represent the current user’s abilities, preferences and requirements. The Context Manager also includes the Sensor Data Manager which aggregates sensor information achieved from the client with already available data in order to update the user profile accordingly at runtime. It provides access to the user profile also for other components of the MyUI system.

All components are implemented in separate modules and use open interfaces so that the system can easily be extended by additional end-devices, context sensors and design patterns.
3.2 Plugin approach of the MyUI infrastructure

After explaining the role of the adaptation engine within the MyUI System and the underlying system architecture, this section gives a technical overview of the concept and implementation of the Adaptation Engine. In addition the plugin approach which is provided by the used CakePHP Framework in combination with the MyUI Middleware is also introduced in this chapter.

CakePHP provides a plugin approach which helps developers to easily extend their applications with reusable software components (plugins). A CakePHP plugin contains its own models, views and controllers as well as its own webroot folder where images, css files, javascript libraries etc. are located. Each CakePHP plugin is saved in the plugins folder of the main CakePHP application. The general folder structure of a CakePHP plugin looks like follows:

```
plugin_name/
    controllers/
    components/
    models/
    views/
    elements/
    helpers/
    webroot/
```

The MyUI Core is implemented and built up as such a CakePHP plugin with a folder structure following the CakePHP plugin structure. The controller package contains the controller classes such as the profiles_controller and applications_controller together with the individualization_patterns component. Classes representing real objects are stored in the models folder. Unlike normal CakePHP plugins the MyUI Core plugin does not contain fully implemented views in its views directory. Instead it makes heavy use of the CakePHP elements mechanism to generate the user interface structures out of interaction patterns and user interface elements at runtime. Besides that, plugins can integrate own external content or libraries such as images, javascript files, css files etc in the webroot folder. In the MyUI Core plugin the adaptation engine itself and the adaptation handlers are located here. The complete structure of the plugin is shown in Figure 3.
4. **MyUI Adaptation Engine in the three-stage process of UI generation and adaptation**

As described in D2.2, MyUI adaptive user interfaces are generated and adapted in a three-stage process (see Figure 4, cf. Peissner et al., 2011). Whereas D2.2 explains the concepts and mechanisms to perform the engaged transformations from a sequence of profiles (models) over user interface building block to complete user interface, this deliverable provides a brief overview of the implementation in the adaptation engine.
4.1 User Interface Parameterization

4.1.1 Summary of the UI Parameterization concept

The MyUI User Interface Profile (UIP) is the result (output) of UI Parameterization. During this first step of user interface generation, general characteristics of the user interface are defined. The UIP includes variables such as e.g. bodyTextFontSize, displayMode (with values from “text only” over “mainly text” etc. to “graphics only”) and voiceInput (on/off). The UIP settings are valid throughout the entire interface and in all interaction situations. User interface parameterization requires information from three input sources:

- Device Profile (DP) with information about the currently available and used I/O devices.
- User Profile (UP) with information about the user and the current environment.
- Customization Profile (CP) with settings defined by the developer to customize a MyUI application.

During UI parameterization, these profiles are transformed into the UIP. A first instance of the UIP is created at the beginning of a new interaction session with a MyUI application. When the available information base about user, environment and devices changes significantly, a repeated user interface parameterization is triggered and the UIP is updated (see Figure 5).
4.1.2 Implementation: profiles_controller

UI Parameterization is triggered when changes in the user profile (UP) occur. The Adaptation Engine requests the profiles_controller every 3 seconds (see Code 1) to supply a current UIP.

```javascript
function fetchUserInterfaceProfile() {
  // Fetch current user profile
  $.ajax({
    type: "GET",
    url: WEB_ROOT + 'myui/profiles/currentUiProfile,
    dataType: 'json',
    async: true,
    cache: false,
    timeout: 50000, /* Timeout in ms */
    success: checkUserInterfaceProfileForAdaption,
  });
}
```

Code 1 fetchUserInterfaceProfile function in /plugins/myui/webroot/js/adaption_engine.js

The currentUiProfile function of the profiles_controller manages the overall process of deriving a UIP from a loaded DP and a current UP fetched from the Context Manager using the individualization patterns and merges the resulting UIP with the CP (see Code 2). The final UIP is send back to the requesting Adaptation Engine encoded as JSON for easy client processing.
The profiles_controller first triggers the XMLRPC call to the Context Manager to get the current UP as an XML structure and converts it into a PHP array structure for further processing (see Code 3).

```php
public function currentUiProfile() {
    $uiProfile = array();
    $rfid = $this->Session->read('rfid');

    // Fetch user profile from context manager and derive ui profile
    $userProfile = $this->getUserProfileFromContextManager($rfid);
    $deviceProfile = $this->Session->read('deviceprofile');
    $uiProfile = $this->IndividualizationPatterns->apply($userProfile, $deviceProfile);
    $uiProfile = array_merge($uiProfile, $this->getCustomizationProfile());

    // Render as JSON
    $this->set('data', $uiProfile);
    $this->header('Content-Type: application/json');
    $this->render(null, 'ajax', '..' . DS . 'json');
}
```

**Code 2 currentUiProfile function in /plugins/myui/controllers/profiles_controller.php**

Together with the DP loaded at the first login of the user from a certain device the profiles_controller uses the individualization_patterns component (see Code 4) to generate and update the UIP. The later class is an essential software component of the MyUI Core because it contains the set of rules of the individualization patterns that are executed to transform the UP to a UIP.
4.2 **User Interface Preparation**

4.2.1 Summary of the UI Preparation concept

During UI Preparation, the most suitable user interface components and elements for the current situation are selected. This step requires input from:

- Abstract Application Interaction Model (AAIM): definition of active interaction situations for each application state
- UIP: reflecting the requirements from the current user, environment and device setup.

UI preparation is triggered when a new state in the AAIM is entered or when changes in the UIP are recognized (see Figure 6).

```php
public function apply($userProfile, $deviceProfile) {
    $uiProfile = array();
    // Apply patterns
    $uiProfile = array_merge($uiProfile, $this->applyFontSize($userProfile, $deviceProfile));
    $uiProfile = array_merge($uiProfile, $this->applyScreenComplexity($userProfile, $deviceProfile));
    $uiProfile = array_merge($uiProfile, $this->applyDisplayMode($userProfile, $deviceProfile));
    $uiProfile = array_merge($uiProfile, $this->applyNavigateAndSelectRC($userProfile, $deviceProfile));
    return $uiProfile;
}
```

Code 4 apply function in /plugins/myui/controllers/components/individualization_patterns.php
4.2.2 Implementation: prepareAdaptation

After receiving the UIP from the uiprofiles_controller, the Adaptation Engine checks if there are any changes between the currently used UIP and the newly received UIP. If the engine detects any changes it starts the UI Preparation and pauses the request for new UIPs. The engine iterates through all registered user interface components and elements and executes their prepareAdaption function which is implemented in their corresponding adaption_handler java script file. Based on the results the Adaptation Engine decides which Adaptation Dialog Pattern is to be executed.

The prepareAdaptation function of the adaptation handlers receives the newly transformed UIP and evaluates which Adaptation Pattern would be most appropriate for the specific situation. Code 5 illustrates how the adaptation handler for the Main Menu decides if it requests to carry out the adaptation dialogue pattern “Explicit Confirmation before Adaptation (EB)” (pre-confirm) depending on if the main menu pattern (number of columns and rows) has to be changed.
User Interface Generation and Adaptation

4.3.1 Summary of the UI Generation and Adaptation concept

A final user interface is rendered by composing the selected user interface components. The generated and adapted user interface complies with the currently available knowledge about the individual user and context requirements at any time during the interaction. Run-time adaptations can be regarded as a repeated user interface generation with mechanisms to switch from one UI instance to another UI instance. This last step of the generation and adaptation process includes three major activities during run-time:

- **UI generation**: A complete UI is created and rendered from the selected components at the beginning of a new interaction session.

- **Profile updates**: During the interaction, relevant interaction events are fed back to the Context Manager to update the UP and the DP. Profile changes trigger a repeated start of the three-stage adaptation process.

- **UI adaptations during use**: When new user interface components and elements have been selected in a repeated user interface preparation process, UI adaptations are carried out.

```javascript
this.prepareAdaptation = function(toProfile) {
    // Determine main menu pattern to adaptate to
    if (toProfile.maxElementsPerScreen >= 16
        && toProfile.numericNavigation == false
        && checkTileSizeForContents(4, 4, toProfile)) {
        // selected pattern: Main Menu - 4x4-Tile Style
        preparedColumns = 4;
        preparedRows = 4;
    } else if (toProfile.maxElementsPerScreen >= 9
        && toProfile.numericNavigation == false
        && checkTileSizeForContents(3, 3, toProfile)) {
        // selected pattern: Main Menu - 3x3-Tile Style
        preparedColumns = 3;
        preparedRows = 3;
    } else if (toProfile.maxElementsPerScreen >= 4
        && checkTileSizeForContents(2, 2, toProfile)) {
        // selected pattern: Main Menu - 2x2-Tile Style
        preparedColumns = 2;
        preparedRows = 2;
    } else {
        // selected pattern: Main Menu - List Style (with 3 elements max)
        preparedColumns = 1;
        preparedRows = 3;
    }

    var menuPattern = columns != preparedColumns || rows != preparedRows ?
        AdaptationHandler.AdaptationPattern.PRE_CONFIRM :
        AdaptationHandler.AdaptationPattern.SILENT;

    return menuPattern;
};
```

Code 5 prepareAdaptation function in
/plugins/myui/webroot/js/handlers/main_menu_adaption_handler.js
4.3.2 Implementation: performAdaptation

After prepareAdaptation (see section 4.2.2 above), performAdaptation is the second important method of the adaptation process. This function executes the entire adaptation for the corresponding interaction situation based on the decisions made in prepareAdaptation (see Code 6).

The performAdaptation function calls a completeCallback to inform the Adaptation Engine that it has finished adaptations and now represents the new variant of the user interface according to the new UIP. In consequence the Adaptation Engine restarts fetching UIP updates.

```javascript
this.performAdaptation = function(toProfile, completeCallback) {
    // Perform adaptations
    if (columns !== preparedColumns || rows !== preparedRows) {
        columns = preparedColumns;
        rows = preparedRows;
        offset = 0;

        layoutTiles(columns, rows, function() {
            performButtonAdaptation(toProfile, function() {
                currentProfile = toProfile;
                completeCallback(thisHandler);
            });
        });
    }
};
```

Code 6 performAdaptation function in /plugins/myui/webroot/js/handlers/main_menu_adaption_handler.js

---

**Figure 7** MyUI user interface generation and adaptation from selected components

---
5. Implementation of basic MyUI concepts

5.1 Interaction Situations

Interaction situations are a major concept in the MyUI approach to adaptive user interfaces. Interaction situations represent the interaction options a user is provided at a certain point in the application, e.g. viewing a list of items, triggering certain functions, going to another place in the application, etc. Each interaction situation is associated with a bundle of Interaction Patterns suitable for the respective situation. Thus, for one situation of an application different interaction patterns provide different user interface solutions which will cover different user and environmental needs and requirements (cf. D2.2). Specifying the interaction situations for all the states of an application is one of the major tasks of a MyUI application developer.

The following example illustrates how the interaction situations and relevant parameters are defined for the weather demo application in the action_forecast method:

```php
public function action_forecast() {
    $this->select_functions = array(
        'Choose City' => array(
            'reference' => array('action' => 'selectLocation'),
            'icon' => array('plugin' => 'weather', 'select_location.png')
        ),
    );

    $this->primary_is = 'view_group_of_items_with_attributes';
    $this->primary_is_params = array(
        'stereotype' => 'weather',
        'data' => $this->getWeatherData()
    );
}

Code 7 Example for defining the Interaction Situation in the application controller

5.2 Interaction Patterns and User Interface Elements

Interaction Patterns and User Interface Elements are implemented as CakePHP plugin elements and stored in the elements directory under the path /plugins/myui/views/elements. The Interaction Patterns and User Interface Elements are realized in a combination of HTML, PHP and javascript. Each Interaction Pattern consists of a ctp view element file and an associated handler that is implemented in javascript (see section 5.3 below). The main menu view file mainly consists of the following code:
// Create menu button for each entry
$menuButtons = '';
$itemCount = count($data);
for ($i = 0; $i < $itemCount; $i++) {
    // Extract element's data structure
    $element = $data[$i];

    $reference = array(
        'plugin' => $element['pluginAttribute'],
        'controller' => $element['controllerAttribute'],
        'action' => $element['actionAttribute'] != null ? $element['actionAttribute'] : 'index');

    // Create menu button
    $menuButtons .= $this->Myui->renderUiElementBundle('menu_button', array('label' => $element['labelAttribute'], 'icon' => $element['iconAttribute'], 'reference' => $reference));
}

Code 8 Generation of buttons in
/plugins/myui/views/elements/interaction_pattern_bundles/main_menu.ctp

It uses the MyUI Helper to render User Interface Elements. In case of the main menu the Interaction Patterns view element asks the helper to render a menu_button element for each menu item representing an available application. This in turn uses the elements contained in the /plugins/myui/views/elements/ui_element_bundles folder of the MyUI Core plugin. Code 9 shows the code generating the HTML structure for a single menu button.
5.3 Interaction Patterns and Adaptation Handlers

All Interaction Patterns and User Interface Elements need to be registered at the Adaptation Engine. For this purpose, each Interaction Pattern includes its corresponding adaptation handler implemented in Javascript. This adaptation handler is initialized with a HTML element representing the rendered interaction pattern and registered at the central instance of the Adaptation Engine (see Code 10).

```php
/* * Create and register adaptation handler */
$this->Myui->requireHandler('TitleBarAdaptationHandler');
$this->Myui->appendDisplayContent( 'titleBarArea',
    $this->Javascript->codeBlock( 'var handler' . $id . ' = new TitleBarAdaptationHandler($("\#' . $id . '"));
    engine.register(handler' . $id . ');');
);
```

The different handlers contain the adaptation logic for the interaction patterns. The handlers are implemented in Javascript and are stored in the directory /plugins/myui/webroot/js/handlers. Each interaction pattern or user interface element has its own handler, which extends the superclass AdaptationHandler. This abstract parent class contains two functions (prepareAdaption and
performAdaption) that need to be implemented by the concrete element handler. The handler functions are executed by the adaptation engine after detecting a UP update and after getting the transformed UIP handed over (see sections 4.2.2 and 4.3.2). In the following example a structural extract of the concrete implementation of the title_bar_adaptation_handler is shown:

```javascript
PreviewAdaptationHandler.prototype = new AdaptationHandler;

function PreviewAdaptationHandler(baseDiv) {
    this.prepareAdaptation = function(toProfile) {
        ...
    };

    this.performAdaptation = function(toProfile, completeCallback) {
        ...
    };
}
```

Code 11 Adaptation Handler initialization in /plugins/myui/views/elements/interaction_pattern_bundles/title_bar.ctp

5.4 Adaptation patterns and undo adaptations

Assuring high levels of transparency and user controllability during system-initiated adaptations is a major issue in the MyUI project. The MyUI adaptation patterns have been described in deliverable D2.2 and in a recent paper (Peissner et al., 2012a). During the last six months of the project the concepts for the adaptation patterns have been implemented in the Adaptation Engine. Especially, the following three major improvements have been made:

5.4.1 Context-specific selection of the appropriate adaptation pattern

Four adaptation dialogue patterns have been implemented:

- Automatic adaptation without adaptation dialogue (baseline)
- Automatic adaptation with implicit confirmation (AI)
- Explicit confirmation before adaptation (EB)
- Explicit confirmation after adaptation (EB)

In empirical studies with end users these patterns have been examined in terms of effectiveness and acceptability. The study results suggest that the pattern AI works best in situations when adaptations can be assumed to result in high benefits and low disadvantages for the end user. This will be the case, when the adaptation includes only minor changes and no switching to other interaction patterns (i.e. the currently presented controls and widgets stay the same while presentation or input modes change, e.g. increased font size, add numeric navigation opportunities, etc.). However, other adaptations causing more substantial changes, e.g. substituting one user interface component by another, will trigger the explicit adaptation dialogue pattern EB.

For the implementation of this principle, the above described adaptation handlers play a major role. For each of the currently displayed interaction patterns an adaptation handler is active. An adaptation handler manages the currently active interaction pattern and all possible variants of the active pattern, i.e. for each bundle of interaction patterns exactly one adaptation handler exists. During the interaction, the adaptation handlers send a notification to the adaptation engine when another interaction pattern of its bundle shall be activated. If the adaptation engine receives only one or more notifications about changes in the current bundles, then the explicit adaptation pattern EB is triggered instead of the implicit pattern AI.
5.4.2 Help the end user to anticipate the effects of adaptations

The second improvement aims at helping the end user to make a deliberate decision for or against a suggested adaptation in the adaptation pattern “Explicit confirmation before Adaptation (EB)”. User can only feel in control of automatic adaptations if they have the feeling of knowing the available options of their decisions.

In an earlier concept version, this feature has been realised by a comparison of the current user interface with a preview of the user interface after adaptation (see Figure 8).

![Figure 8 Adaptation dialogue pattern “Explicit confirmation before adaptation (EB)” with preview of the adapted user interface (UI concept sketch)](image)

A technical implementation of such dynamic preview, however, turned out to very difficult. A first implementation relied on a preview with two iFrames HTML elements which load to different variants (current and intended) of the current page as content. A first problem with this solution is the fact that the comparison of the current UI with the preview would require extracting the most informative screen part and scaling it to a perceivable size to be displayed in the iFrames. A second problem is that the intended comparison would require a representation of two variants of the user interface with two different instances of the UIP which need to be valid at the same time. A solution to both problems would mean an enormous implementation effort while the benefits for the overall project are rated only mediocre.

Therefore, our current solution concentrates on generating an informative formulation of the confirmation request in the message box. Our assumption is that a wording which provides the user with explicit clues to the most important changes. The new implementation relies on checking the UIP for relevant changes which lead to interaction pattern adaptations. Currently, all instances of switching to other interaction patterns can be fully described by increased/decreased font sizes with less elements on the screen and fewer/more screen elements (as a consequence of UP changes in the variable attention). Therefore, a prioritization of these two cases is sufficient for generating an instructive confirmation request. The formulation rules can be summarized as follows:

If
  UIP variable maxElementsPerScreen has changed,
Then
  ask: “Would you prefer a simpler screen layout with fewer elements?” | “Would you prefer a more complex screen layout with more elements?”
Else,
  if
    one of the UIP variables for fontSize has changed,
Then
  ask: “Would you prefer bigger text and graphic elements?” | „Would you prefer smaller text and graphic elements?”
5.4.3 Undo the latest adaptation

The undo-function is an important feature for the controllability of automatic adaptations. The implementation of the MyUI Adaptation Engine offers the option to undo the latest adaptation. This is especially important for the proper realization of the implicit adaptation pattern AI, as in this case the undo is the only available mechanism to influence the adaptation behaviour. Also after explicitly accepting an adaptation in the adaptation pattern EB the user might change her mind when experiencing the new UI.

For the proper implementation of the undo function it is necessary that the Adaptation Engine sends an event to the Context Manager whenever an adaptation is carried out. This event is implemented in the method detectedAdaptation(), which triggers another method (users/detectedAdaptation) on the server by sending an Ajax call. The users/detectedAdaptation method in turn sends a XMLRPC message to the Context Manager. Then, the Context Manager saves an image of the user profile, to which it can return after another adaptation. The user can trigger the undo function by pushing the “Revert” button in the adaptation menu. The undo function is implemented in the method revertAdaptation in the AdaptionControlAdaptionHandler (app/plugins/myui/webroot/js/adaptioncontrol_adaption_handler.js).

5.5 Main functions of the adaptation process

The Adaptation Engine is the central unit of the three-stage process user interface generation and adaptation process. This unit registers all adaptation handlers which should be notified if any changes in the UIP take place. The Adaptation Engine triggers the UI Parameterization process in which the context data are collected. When changes between the previous and the new context data have been recognized, the Adaptation Engine starts UI Preparation (prepareAdaptation) and UI Generation and Adaptation (performAdaptation) (see Figure 9).

![Figure 9 Main functions and concepts of the MyUI adaptation process](image)

The adaptation process is implemented as a JavaScript file. The flow chart diagram in figure 9 illustrates the process and the involved functions. It shows also the allocation of tasks between the
different components and the caller/callee dependencies. The seven major functions of the adaptation process are explained in the following:

**Register:** The element interaction_pattern is responsible for positioning the interaction patterns and user interface element on the screen. It registers the associated handlers at the adaptation engine. If the engine is already running and has a current UIP, the MyUI element handled by the adaptation handler will be adapted to the existing UI profile. Otherwise, it will be adapted as soon as a new UIP is available.

**Run:** The function Run detects UIP changes and manages the adaptation of registered MyUI elements. This function is executed after the website has been rendered completely.

**FetchUserInterfaceProfile:** This function fetches the UIP from the server and forwards it to the checkUserInterfaceProfileForAdaption method for further management. This is realized with an Ajax call that requests the UIP via the users_controller from the MyUI Middleware.

**CheckUserInterfaceProfileForAdaption:** This function is executed as a callback if the Ajax call returns the UIP successfully. Then it checks if the new profile differs from the current profile and manages the adaptation of the registered MyUI elements to this new profile.

**PrepareAdaption:** This function prepares the adaptation of the user interface. Each registered MyUI element owns a prepareAdaptation method. All prepareAdaption methods of all registered MyUI elements will be called. Based on the returned values from the requested Interaction Patterns the engine will then select the appropriate Adaptation Patterns.

**BeforeAdaptation:** This function executes a pre-adaptation transition pattern based on the decisions of the Interaction Patterns and User Interface elements during the adaptation preparation process. Depending on the selected adaptation pattern and the user behavior this function executes or declines the adaptation of the user interface.

**PerformAdaption:** This function performs the actual adaptation of the user interface. It calls the performAdaption method of all registered MyUI elements and monitors the completion of the adaptation with corresponding callbacks. The adaptionCompleted method will be called when all registered MyUI elements have reported their adaptation as completed.

**AfterAdaptation:** This function executes a post-adaptation transition pattern based on the decisions of the Interaction Patterns and User Interface elements during the adaptation preparation process. Depending on the selected adaptation pattern and the user behavior this function commits or undoes the adaptation of the user interface. Additionally, the current UIP is overwritten by the newly fetched UIP. Then the adaptation engine restarts its cycling adaptation process by fetching the UIP from the MyUI Middleware again.
6. MyUI Adaptation Engine in action – example use cases

This chapter illustrates the adaptation effects the MyUI Adaptation Engine is able to produce by means of some example use cases. The shown user interfaces are generated by the Adaptation Engine using the corresponding Interaction Patterns selected from the pattern repository.

6.1 Main Menu

The following figures (Figure 11 to Figure 13) illustrate some different variants of the MyUI Main Menu:

- Figure 11: Main Menu for a “standard” user without special needs, i.e. all capability-related UP variables set to 0.
- Figure 12: Main Menu adapted for a user with a highly reduced ability in language reception.
- Figure 13: Confirmation dialog (adaptation pattern EB) when adapting the Main Menu caused by an improved value of the UP variable attention.
Figure 11 Basic 4x4 variant of the main menu with preview

Figure 12 Icon only variant of the main menu in 3x3 style
6.2 Weather Application

The following figures (Figure 14 and Figure 15) illustrate two different variants of the MyUI Weather Application forecast screen:

- Figure 14: Forecast screen adapted to a user with a reduced ICT literacy which results in the activation of numeric navigation.
- Figure 15: Forecast screen with a reduced screen complexity adapted to a user with a reduced attention value in the UP.

Figure 14 Basic variant of the weather forecast display with numeric navigation activated

Figure 13 Confirmation dialog when increasing the screen complexity
6.3 User access to the user interface profile

Figure 16 shows an example of the profile settings menu where the user has direct access to UIP variables font size, display mode and screen complexity via shortcuts.
6.4 Setting up the MyUI runtime environment

To be able to run the Adaptation Engine it is necessary to set up a MyUI runtime environment. It consists of the following two software components which need to be installed on the runtime device:

- MyUI framework and website as CakePHP application
- MyUI Context Manager Infrastructure (client and service) as C# application

The source code of the entire MyUI infrastructure is saved in a SVN repository. In order to run the system the following software components are required:

- XAMPP (Apache, MySQL) for the MyUI website
- .Net Framework for the Context Manager (Visual Studio Express IDE works best as it includes all required libraries: http://www.microsoft.com/germany/express/)
- Eclipse with Aptana, Subclipse, Web Development, PHP plugin

6.4.1 MyUI Core and website

After successfully installing XAMPP a folder can be created under the htdocs path in which the MyUI project can be checked out. Subclipse offers the possibility to check out the project in a wizard-guided style and to compile a PHP project in eclipse at the same time.

After defining the repository location (Figure 17) the project can be downloaded via the checkout wizard. At the same time a new project can be created by selecting “PHP project” from the “create new project wizard” menu. Now the project path needs to be set to the HTDOCS folder of the XAMPP Apache (instead of the default location) in order to make the code available also for the browser. The website can only be started with the context manager infrastructure installed.

![Figure 17 Add new SVN repository and define its location](image)

6.4.2 Context Manager Infrastructure

The Context Manager (CM) consists of two components: the Client and the Service. The CM Client is sort of remote application to manipulate the user data in the CM. The CM runs as a Windows service and is auto-started together with windows. A complete installation manual can be found in the checked-out folder under /MyUI/contextmanager/doc/Context Manager
Documentation.docx. It is important to follow all instructions step-by-step (CMD to be opened as admin, service installation, create and activate database, etc.) so that the CM can work correctly.

After installing the service, the database can be activated and started with the aid of the CM client to be found under /MyUI/contextmanager/bin/v2.1/FZI.MyUI.Client.exe. The, UP variables can be manipulated via the user interface of the User Manager.

![Activate the database in the CM client](image.png)

**Figure 18 Activate the database in the CM client**

### 7. Conclusion

The here documented MyUI Adaptation Engine implements the MyUI adaptation concept as described in deliverable D2.2. The main capabilities of the adaptation engine can be summarized by the following functions:

- Generate a user interface from reusable software components (represented by MyUI design patterns) according to the MyUI context model.
- Adapt the user interface during run-time and during the interaction on the basis of newly available knowledge about the user and the context of use.
- Maintain a close connection to the MyUI User and Context Management Infrastructure in order to receive profile updates and to notify the context manager about relevant user interface events.
- Provide mechanisms which help the end user to understand and control the system’s adaptation behaviour and to configure relevant profiles.

A technical validation of the adaptation engine has been carried with a couple of demo applications, e.g. main menu for service selection, email service, weather app. “The functional tests yielded satisfying results. At any time, the MyUI system generated a meaningful and consistent user interface. Relevant updates in the user and context profile during the interaction always resulted in the expected run-time adaptations. The recorded system performance for user interface generation and adaptations during run-time was excellent. The time needed to generate a new instance of the user interface was equal to the (very short) time for loading a new page of the web application without adaptations.” (Peissner et al., 2012a)

After a quick start for the requested (by the reviewers) month-12-live-demo, reimplemention of major parts of the adaptation engine has been necessary in order to fully satisfy MVC compliance and to enable the generation of adaptive user interfaces on the basis of an Abstract Application Interaction Model (AAIM) as created with the aid of the MyUI development toolkit (WP 3).
References


