Experiences in Using Patterns to Support Process Experts in Wizard Creation

Birgit Zimmermann[1, 2], Christoph Rensing[1], Ralf Steinmetz[1]

[1] KOM Multimedia Communications Lab
Technische Universität Darmstadt
Merckstr. 25, 64283 Darmstadt, Germany
{birgit.zimmermann, christoph.rensing, ralf.steinmetz}@kom.tu-darmstadt.de

[2] SAP AG
SAP Research CEC Darmstadt
Bleichstr. 8, 64283 Darmstadt, Germany
birgit.zimmermann@sap.com

1 Overview

Users of software often complain that many software solutions only insufficiently support them in solving their problems and performing their tasks. This phenomenon occurs with all kinds of software. It can also be seen with tools that are especially developed to support users in performing specific tasks. Working on a tool supporting users in performing so called adaptation processes we found the same problem. (These processes are needed to adapt existing E-Learning material in order to make it suited for changed usage scenarios.) To offer tool support for adaptation processes we created a wizard based on a pattern based description formalism for adaptation processes. This paper presents our experiences with this approach. The next chapter gives an introduction to adaptation processes.

2 An Introduction to Adaptation Processes

Creating high-quality E-Learning material is a time and cost consuming task. Re-using existing material could reduce these costs. But often a one-to-one reuse of the existing material is not possible, as the new scenario of usage differs to a certain degree from the original usage scenario. Therefore, to achieve a high quality, it is necessary to adapt the existing material to the new usage scenario.

The processes needed to perform the adaptations are structured hierarchically: A process is performed by executing several process steps. These process steps can consist of smaller process steps or of atomic operations that cannot be split up into smaller units (see Figure 1 on the next page).

Let us have a look at one example process: The adaptation of E-Learning material to a changed (corporate) design consists of several process steps like exchanging logos and images

Proceedings of the 13th European Conference on Pattern Languages of Programs (EuroPLoP 2008), edited by Till Schümmer and Allan Kelly, ISSN 1613-0073 <issn-1613-0073.html>.

that do not fit, or changing fonts, backgrounds, colours, etc. The process step ”exchanging images” consists of several atomic operations like identifying all used images, testing for each image, if it fits to the requirements, finding images that have to be used instead of non-fitting images etc.

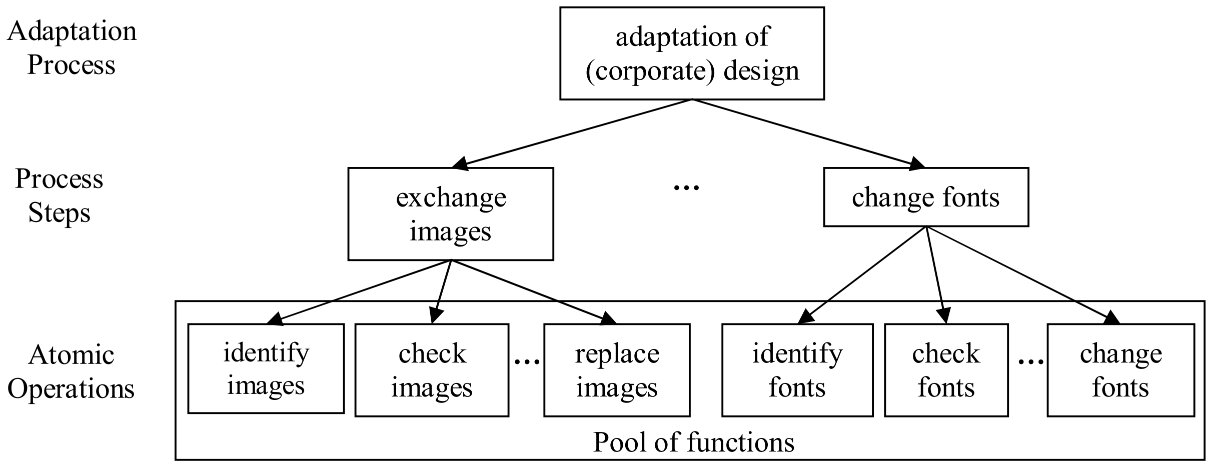


Figure 1: Process hierarchy [18].

In many cases it is necessary to perform several adaptations in order to achieve a good result. When changing the layout to make it suited for a different corporate design, it is for example often also needed to change the terminology, in order to adapt it to the terminology of the new company. But in reality, often the persons, who have to adapt the material, are not experts in performing all needed tasks. Mostly they only have a certain basic knowledge of how to perform the adaptation processes. This means that compared to Dreyfus’ model of skills [4] they are novices in performing adaptation processes. But with the help of a tool that is based on the knowledge of persons from the expert level it is possible that the novices are supported in performing adaptation processes in such a way that they are enabled to achieve results that a user from a higher skill level would achieve. To develop such a tool support was our goal within the Content Sharing Project (http://www.contentsharing.com).

To be able to offer support for adaptation processes it was necessary to find out how the adaptation processes have to be performed and how a useful tool support could look like. Therefore we carried out a survey of persons being experts in performing the processes needed to adapt existing E-Learning material in order to make it suited for changed usage scenarios [18]. These persons perform adaptation processes very often. Therefore they can offer a detailed description of how to perform the processes.

Many of the adaptation process experts mentioned that existing tools are not well suited to support users in performing the adaptation processes. Especially novice users are often not able to use these tools. The main reason for this problem is that the tools often do not represent the processes themselves. They are based on the software designers understanding of the processes [9]. But in many cases this understanding differs from the processes as they are performed and understood by the process experts [11]. Thus we decided to look for a possibility that allows adaptation process experts to be involved more directly in the development of a tool offering support for adaptation processes.

3 Problem: How to Enable Process Experts to Describe their Knowledge of Adaptation Processes?

Traditional software development often starts with analysis activities in order to collect information about how processes are performed. The requirements from user side as well as from system side are collected. Based on these requirements, models of the processes are developed and implemented [12]. The exact proceeding can deviate depending on the underlying project structure (e.g. software development according to v-model, spiral models, extreme programming, rational unified process etc.).

But this proceeding often leads to problems caused by misunderstandings between process experts and software designers and developers [11]. To make things worse, it is often not possible for process experts to control if the models really describe their processes, as they do not have knowledge of common modelling formalisms like UML or ARIS. According to Siau et al. [16] UML is too complex in many cases and its constructs are ambiguous. The same holds for ARIS [13].

Someone who for example is working as a translator (an adaptation that occurs very often) is a process expert for translation. Normally this person is not simultaneously an experienced software developer or designer. Therefore most translators are not able to deal with common modelling formalisms as UML, ARIS or BPMN [1], as they do not need these methods in their daily work. The same holds for many other adaptation processes: Mostly persons have expertise in the processes they deal with in their daily work. Thus they are able to give a detailed description of how they perform these processes. But many of these persons have never learned to describe their processes with common modelling formalisms.

Thus it would be desirable to facilitate adaptation process experts to describe their process knowledge and to use this knowledge for software creation in a way that is easy to learn and easy to understand and that does not require extensive training. In order to solve this problem it was necessary to find a possibility that fulfils the following requirements:

- It must be possible for adaptation process experts to describe their knowledge with an easy to use formalism. In addition the process experts should be enabled to create a prototypic support tool based on the created descriptions. This prototype should give process experts a possibility to find out if a given process description is correctly reflected in the prototype and if a tool based on the prototype would support the described adaptation process in the desired way.

- At the same time it has to be considered that the description formalism is structured in such a way, that on one hand it is possible to create the prototype mentioned above based on the description in a way that does not require development skills. On the other hand, the prototype has to solve as a basis for further development. This means that the prototype as well as the process description have to contain all information needed to create a prototype and to allow a developer to implement a computer program by enhancing the prototype.

For the adaptation process expert it is important that the description formalism can be used without extensive training. For the developer it is important that the description formalism allows using familiar software development proceedings and tools as far as possible. We wanted to find an approach that meets both requirements. This approach should be used to enhance the software development processes existing by now. It was not intended to replace those processes.

4 Solution: Pattern Based Process Description

Patterns document proven solutions to recurring problems; they describe best practices [8]. Patterns are noted in natural language. Hence they are easy to understand for the persons of whom knowledge is collected with the patterns [7].

Patterns offer a regular form and they can have a structured, fixed notation. They can be stored in an XML format as the Pattern Language Markup Language (PLML) [10]. PLML is an XML DTD, which originally was thought of as a common standard for HCI patterns. XML is very flexible. It offers the possibility to be used for several different areas of application. For example, XML can be rendered to HTML or other formats, e.g. by using XSLT. In this way it is readable for a non IT person. At the same time it is very structured and thus machine readable. Besides it can be imported to established UML modelling tools by using XML Metadata Interchange (XMI).

Patterns are written down in natural language. Therefore pattern based process descriptions are easy to understand for adaptation process experts. In addition software designers as well as developers will find their familiar views on pattern based process descriptions stored in XML, because of the flexibility of XML. Thus we decided to use patterns to capture the process knowledge of adaptation experts and to store these patterns in an XML format. We have written down the outcome of our user survey using patterns to describe the adaptation processes. This led to a couple of initial patterns. We revised the initial patterns together with process experts. In addition we made sure to find at least three known uses for each pattern by searching for successful applications of the patterns. Some of the results have been published at PLoP conferences ([19], [20]).

These patterns describe the processes needed to adapt existing E-Learning material on a high level. They contain important information about how to proceed. Especially they contain a section naming all needed process steps. Some other pattern formats also include sections describing concrete steps in detail. But as process steps are sometimes needed in several processes we name the steps within a pattern, but we separated their concrete description from the patterns. We added a second kind of descriptions for the process steps, called *how-to guides*. These how-to guides explain in detail what has to be done to perform each process step and which smaller process steps or atomic operations are needed during execution. Compared to the patterns describing the whole process the how-to guides are much more detailed. Most atomic operations are used in several process steps. Therefore we also have separately written down instructions for all atomic operations needed in the process steps.

Wizards are a common solution in computer science, to offer users without expert knowledge a step by step guidance through processes. According to [6] wizards can be used, if novices have to perform a complex task composed of several steps. The novices know which goals they want to reach, but they do not necessarily know which steps they have to perform. A wizard helps them in reaching their goals.

Wizards are easy to use even for users who are not familiar with the processes supported by the wizard. Adaptation processes often have to be carried out by persons who are not experts in performing these processes. Therefore they need detailed step by step guidance through the adaptation processes. Thus we decided to develop a wizard as a supporting tool for this kind of processes.

As stated before it was our aim to enable the process experts to be involved more directly into the software creation process in order to achieve software that is based on their knowledge. But we did not want them to learn common modelling formalisms or programming techniques. Thus we were searching for a possibility allowing the process experts to prove the outcome of their process descriptions. As stated before adaptation process experts often have no knowledge of common modelling formalisms. We therefore decided to develop a method that allows process experts to generate a prototypic wizard, which is based on the process descriptions. With the prototype it can be checked, whether the underlying process descriptions really describe how the process is performed. As most of the experts in performing adaptation processes do not have the knowledge to develop such a wizard, the method should be easy to use for persons without programming experience. Therefore we wanted an automated wizard generation that does not require expertise in software development. In addition the wizard should be based on the process descriptions written down in patterns, process step descriptions and atomic operations descriptions, as they contain all information needed to carry out the processes.

To be able to offer this possibility we needed a two-stage proceeding for the creation of process descriptions: First the knowledge of process experts had to be captured in an easy to understand format. In a second step this format had to be mapped to a formal XML representation allowing to generate a wizard based on the given information. Therefore it was necessary to formalize the process descriptions to such a degree that they can be used as basis for automated code generation. In addition we had to find a possibility for an easy to use automated wizard generation. In the following section we show our concept for solving this issue.

5 Implementation

The patterns, process step descriptions and atomic operations descriptions contain the knowledge of the adaptation process experts. To be able to create a wizard out of these process descriptions we needed a structured, machine readable representation of this information. Because of its flexibility we have chosen an XML notation (adapted from PLML) to store the patterns describing the adaptation processes (as described in section 4). We also used XML to store the process step descriptions and the atomic step descriptions.

But most process experts have no knowledge of XML. Because of this we developed a process description input tool (PIT), which supports users in writing down pattern like process descriptions and saving them as XML files. PIT stores the descriptions given by the process experts in two files: one file containing all textual information about the process and a second file containing a graph representation of the structure of the process. This proceeding corresponds to the two-stage proceeding for the creation of process descriptions represented before: In a first step PIT allows to capture the process knowledge in an easy to understand format. In a second step it maps the format to a formal XML representation.

The XML files are taken as input for a wizard generation tool (WGT). WGT generates a wizard by interpreting the data provided by PIT. The generated wizard represents a first prototype of a process support wizard. It can be taken as a starting point for further development.

Thus the pattern-based wizard generation approach proposed in this paper is based on a three-step proceeding:

1. Writing pattern like process descriptions
2. Automated wizard generation
3. Extension of prototypic wizard

Each of these steps is explained in detail in the next sections.

6 Step 1: Writing Process Descriptions

In the first step the process expert has to create a pattern like process description. For each process one pattern is created. If a process is more complicated and contains several sub-processes, it is possible to create patterns for the sub-processes and to link them to the super process. (This is realized via a specific type of related patterns.) In addition the process step descriptions and the descriptions of the atomic operations have to be created.

To support process experts in creating process descriptions in the needed format the process description input tool (PIT) has been created. PIT is a Java application developed under Eclipse. It offers an input form helping process experts to describe processes in the required pattern based notation formalism.

Figure 9 in the appendix presents a part of PIT's input form for process descriptions. One can see that the process description contains common pattern elements like intent, context, or problem statement. These elements provide a reason why a process can be applied, why a certain context has to be fulfilled to be able to perform the process etc. Thus they are useful for other persons who might want to perform the described process. All this information is very valuable to all persons, who have to decide, which process helps them in solving a specific problem, and who are not process experts.

As shown in figure 9, mandatory input fields are marked with an asterisk. We have chosen these elements as mandatory as they are most important to other persons in order to understand why and how a process has to be performed. Mandatory elements are:

- The **pattern ID** is a unique ID used to address the pattern. It is created automatically.
- The **name** provides a first, rough idea what the pattern is about.
- The **problem** section describes the situation addressed by the pattern.
- The **solution** explains to the user, how the problem can be solved.
- The **process steps** list all steps needed to execute the solution.
- The **consequences** help other persons, to decide if they want to apply a pattern or not, depending on if the positive consequences are more important than the negative ones.

The non mandatory elements are needed if a process expert wants to write a pattern, but they are not necessary, to offer a useful process description:

- The process expert has a certain **confidence** in the process description. (As we talk about process experts the confidence should be high.)
- The **intent** gives a short overview what a pattern is about.
- The **context** describes the situation where the pattern can occur.
- The process expert can give an **example** of a successful application of the pattern.
- The **forces** describe the sometimes contradicting trade-offs that must be considered when performing the process.
- **Known uses** describe situations where the pattern has been applied successfully.
- **Related patterns** can exist, but it might be that there are no related patterns.

When the process expert stores a process description, the given information is stored in XML files. The XML files contain in one file the textual description of the process and in a separate file the graph representation defining the process flow and all its dependencies and preconditions. In the appendix a DTD of the XML file containing the process information can be found. Many of the elements of this DTD are taken from PLML [10]. The PLML v1.1 DTD contains some additional elements that have not been taken into account here: `alias`, `synopsis`, `diagram`, `rationale`, `literature`, `pattern-link` and `management`. The patterns taken as a starting point for this work do not need these elements.

But our patterns contain some elements that have a slightly different meaning compared to PLML. The PLML element `illustration` is called `example_illustration`, as to our understanding, this better explains what is meant by this element. To be able to better differentiate the `example` as used in PLML from the `example_illustration` we call it `example_explanation`. What is referenced as an `implementation` in PLML is called `process_steps` in our DTD. The `related_patterns` already contain a link to the pattern they are related to. Therefore our `related_patterns` are a kind of a combination between `related-patterns` and `pattern-link` in PLML.

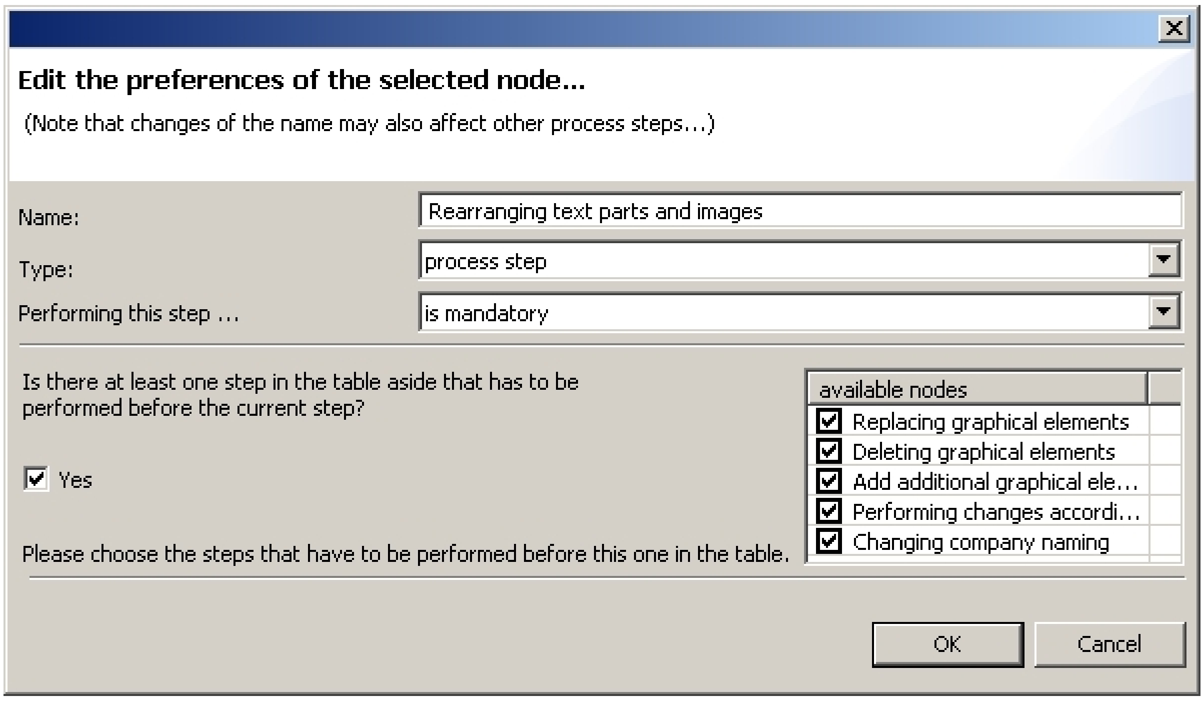
The following elements that are used in our DTD are not mentioned in PLML, but they are necessary in our approach:

- `intent`: The intent explains what the pattern aims for.
- `known_uses`: The known uses are part of the example-section in PLML. In our patterns the example only contains one known-use. Other known uses can be added by using this element.
- `consequences`: The consequences occur when the pattern has been applied. Positive and negative consequences can occur. As for a reader of a pattern it is very important to know, what will happen when applying the pattern, we added this section.

6.1 Defining Process Steps and Atomic Operations

The steps of a process, their interdependencies, and the preconditions for the execution of each step constitute the process. Thus they are essential for the execution of a process. Also branching and cycles within the process flow are of importance. Hence PIT provides a special wizard to define the process steps. This wizard allows a fine granular specification of the process flow without requiring special knowledge of process modelling. It is started by pressing a button to add a step to the process description.

With this wizard users can define for each step if the execution of a step is mandatory or optional, or if certain preconditions have to be fulfilled, before the step can be executed, or if there are dependencies on other steps. Branches are embodied by a special kind of precondition: If the precondition leads to one result, one step is executed, if it leads to another result, another step is executed. Cycles are defined by specifying a step that is the starting point for the cycle and another one, which is the end point. In addition a termination condition has to be defined. This is again modelled by a special kind of a precondition. Figure 2 shows the screen used to define a step that has to be performed, if other steps have been performed before. Another part of the wizard helps users to define preconditions and a third one allows to model cycles. In addition an input form exists, used to describe how a process step has to be performed. The description of process steps is also stored in the process description file.


Edit the preferences of the selected node...

(Note that changes of the name may also affect other process steps...)

Name: Rearranging text parts and images

Type: process step

Performing this step ... is mandatory

Is there at least one step in the table aside that has to be performed before the current step?

✓ Yes

Please choose the steps that have to be performed before this one in the table.

available nodes
✓ Replacing graphical elements
✓ Deleting graphical elements
✓ Add additional graphical ele...
✓ Performing changes accordi...
✓ Changing company naming

Figure 2: Defining dependencies between steps.

Compared to the pattern based description of a whole process, the process step description is very short: It contains the name of the process step, a detailed description of how to perform the process step and a listing of all smaller process steps and atomic operations needed when performing the process step. Sometimes a process step is as complex that it can be regarded as a complete process on its own. Then it is possible to create a process description for this step. This description can be added to the process step description via a special link. In the wizard this is presented as a link to an additional page containing the complex description. Users of the wizard can then read this additional information.

As steps themselves consist of smaller steps - atomic operations or again process steps - it is also possible to define all operations and smaller steps needed to perform a process step.

Again a wizard helps to state if the execution is mandatory or optional, or if certain preconditions have to be fulfilled. Dependencies on other atomic operations can also be determined. For each atomic operation a description has to be provided, how the operation is executed. This can be done via a special input form.

There exist three different kinds of atomic operations: queries, decisions and executions.

- **Queries** are needed to determine information. For example: Find all images used in an E-Learning course.
- **Decisions** are needed if a person, who performs the process, has to decide on something. For example: Decide for each image, if the image has to be deleted or not.
- **Executions** are needed whenever something is changed or done. For example: Delete all images that have been chosen for deletion.

Based on the information represented via the process steps and the atomic operations the second XML file is created. This file represents the process flow. It names all steps and atomic operations. For each step or atomic operation it contains the information whether the execution is mandatory or not, and if there exist preconditions or dependencies. Via the pattern ID, process step ID, and atomic operation ID it is possible to map the information stored in this file to the information stored in the description file. Listing 2 in the appendix contains an example for such a process graph description.

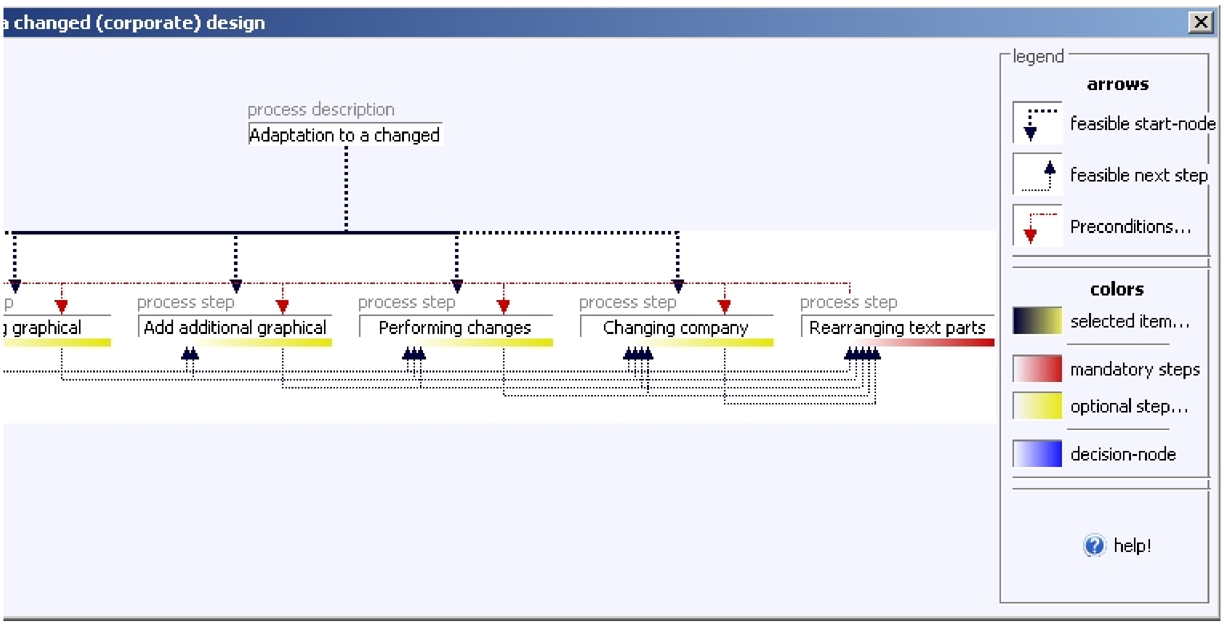


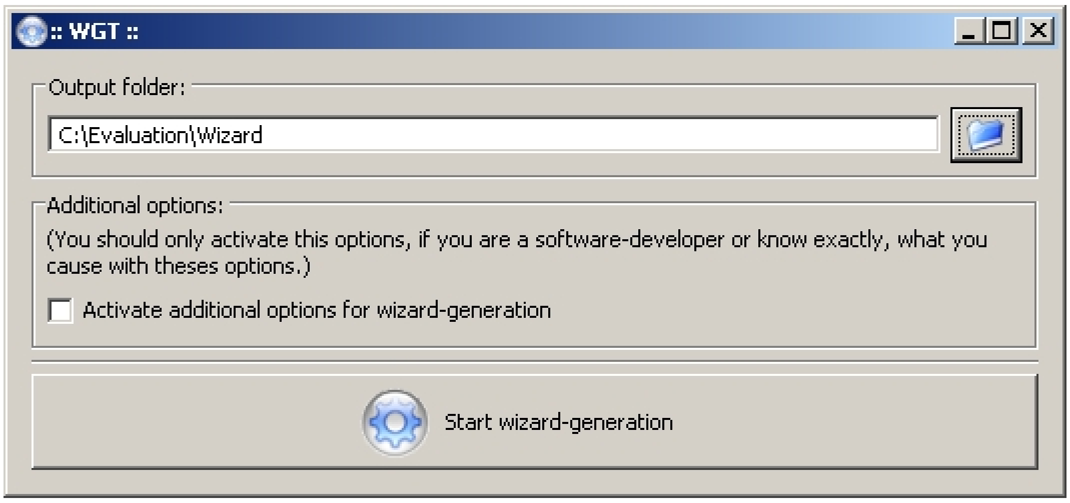
Figure 3: Part of the visualization of a process graph.

PIT offers several possibilities to work with the generated process descriptions: It is certainly possible to edit an existing description. In addition the textual description stored in XML can be rendered as HTML in order to view it via a browser. Thereby it is possible to read the entire process description as a continuous text. Thus it is more comfortable to control whether the description is complete and accurate. Furthermore, a visualization of the process graph is available that represents all steps of the process flow. (Figure 3 shows the first level of the process graph corresponding to listing 2 in the appendix. By clicking on a process step the level under this step opens in the viewer.) At the moment this visualization is only a first prototype. Further enhancements are planned. As the process descriptions are stored in XML it is possible to offer several visualizations that are tailored to the needs of several persons (a developer and a process expert need, for example, different visualizations). In addition it is planned to offer an export to XMI. The XMI files could be used to provide a class diagram and an activity diagram representing the process.

7 Step 2: Automated Wizard Generation

The XML files generated by PIT serve as input for the wizard generation tool (WGT) used in step 2 of the proceeding proposed here. WGT is a Java application developed under Eclipse. Based on the information of PIT's XML files it generates a wizard that, as a first prototype, can be used as a starting point for further development.

WGT is started by selecting a menu entry in PIT. WGT reads the process description that is actually open within PIT. The user specifies where the generated wizard has to be stored and then starts the wizard generation by simply pressing a button (compare figure 4). WGT then parses the XML files created by PIT. It extracts the information contained in the files and fills several predefined code templates with this information. By this procedure all needed Java classes for the prototype are created. The "Activate additional options..." section shown in figure 4 can be activated to open a dialog that allows to specify, how the atomic operations have to be distributed over the wizard pages. (Later in this chapter this is explained more detailed.)


:: WGT ::

Output folder:

C:\Evaluation\Wizard

Additional options:

(You should only activate this options, if you are a software-developer or know exactly, what you cause with theses options.)

☐ Activate additional options for wizard-generation

Start wizard-generation

Figure 4: Screenshot of WGT.

The main purpose of the wizard generation tool is to generate a wizard based on the information given by the process descriptions created with PIT. The wizard should have a graphical user interface, which allows directly after its generation that the process expert evaluates whether the wizard contains all needed information and whether the process flow is correct. The design of the user interface should follow common rules for designing user interfaces, as described in [3], [14].

The atomic operations used in the wizard are not automated directly after generating the wizard. Instead the wizard contains descriptions telling a user how to proceed. We believe that it leads to better results, if an experienced software developer checks, where it is reasonably possible to automate certain operations. Where this is possible the developer can extend the automatically generated source code. To make it easier for the developer, to find where the source code can be extended, comments are added to the automatically generated source code during its creation.

Another important point during wizard generation is the time needed to generate the wizard. As long as the wizard does not fulfil all needs the process expert will change the description created with PIT and generate a new version of the wizard. Thus it is highly probable that the process expert will use WGT several times until a wizard is generated that really fulfils all needs. Therefore WGT has to be fast in order to reduce the time, where the process expert has to wait for the generation to finish.

WGT takes the process description as a model of the process that is supported by the wizard. This kind of software creation is called generative programming [2], which is a special kind of model driven software development (MDSD). There are a lot of different approaches for MDSD [17]. As large parts of the wizard stay the same for all kinds of processes regarded here, we decided to use code generation templates to create the wizard. This allows a fast source code generation.

The wizard generation tool consists of three logical units that are passed one after the other (shown in figure 5):

1. In the first part the XML files provided by PIT are read. The information about the process flow is transferred into an internal process structure model and the information contained in the descriptions is extracted.
2. The second part uses this information to map it to Java code templates. The textual information is used to enhance the content of the graphical user interface. The structural information is used to create all process steps and to transfer the structure of the process flow to the wizard. By this all needed Java classes are generated.
3. The third part writes the generated source code to several Java classes and marks all parts in the source code that can be enhanced by adding additional source code. In addition the created classes are compiled and a batch file is created that allows starting the wizard comfortably.

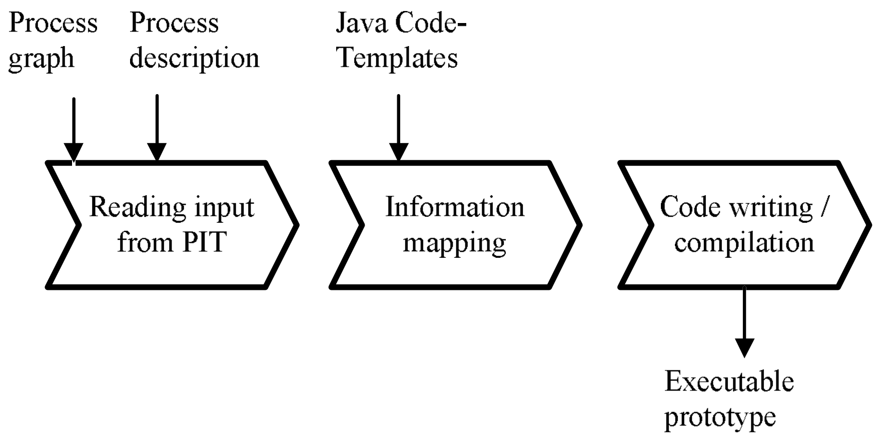


Figure 5: Three phases of wizard generation.

The prototype wizard generated by WGT is based on the model-view-controller principle [5]:

- The process graph containing information about the process flow serves as a *model*.
- The wizard pages providing a graphical user interface build the *view*. Those pages contain the textual information of the pattern-based process description. They are based on Java code templates. WGT instantiates those templates by filling them with the given information. The pages then contain a detailed description on how to perform the process, each process step, and each atomic operation.
- The *controller* of the wizard passes events caused by the user to the model and reactions caused by the model back to the user. The controller interprets the process flow information provided by the model. Depending on the user input the controller monitors which step has to be performed at which time and which step is possible as next step. Thus the controller assures a correct process flow. In addition it contains information about the actual state of the wizard, as it stores, which steps have been carried out so far as well as their results. This is important for a further implementation of the wizard enhancing it with automated functionality. The controller also has to collect and distribute all data that have been created or are required when performing several steps in an automated way.

As large parts of the source code stay the same, code generation templates exist for most wizard classes. There are several kinds of page templates and composite templates depending on the function of each part of the wizard: One template exists for the start page of the wizard, another one for the last page. A special page template exists for process steps. This template contains a part, where the description of the process step can be added. The atomic operations are realized via composites that are based on composite templates. For the three different types of atomic operations three different templates are used. The composites of the operations of a process step are grouped together on one page, which is also created based on a template.

After generating all the Java classes of the wizard, WGT starts to compile those files and to build an executable wizard. A batch file is created that allows to start the wizard comfortably. The executable wizard is a first prototype. It offers the process expert the possibility to prove, whether the wizard represents and supports the described process. If this is not the case, the process expert can refine the process description and generate a revised version of the wizard.

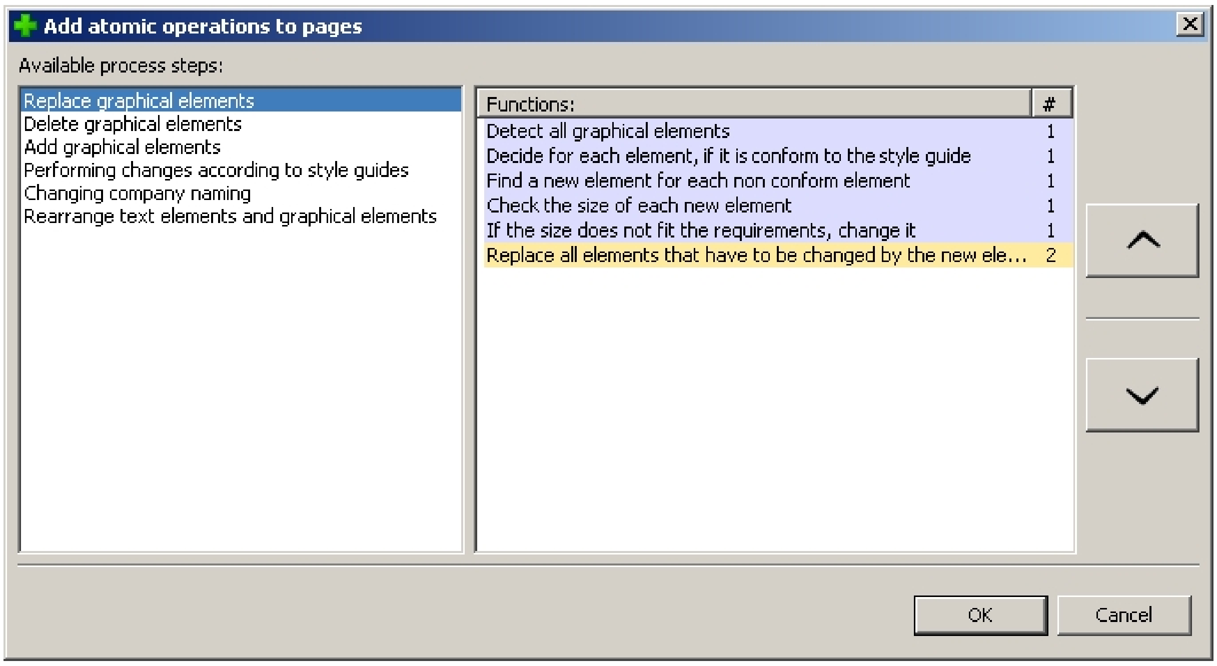
7.1 Arranging Steps and Operations on Pages

One problem when creating the wizard is how to distribute the process steps and atomic operations over the wizard pages. There are several possibilities how to solve this problem: One could create one page explaining each process step and one additional page for each atomic operation. As many adaptations consist of quite a lot of process steps and atomic operations, this can easily lead to a huge number of pages with sometimes only a short explanation on it. Therefore this possibility has been discarded.

Another possibility would be to create one page for each process step, and to add all atomic operations needed in the process step to this page. But many process steps contain ten and more atomic operations. Then the pages would become huge and overcrowded with information. We also discarded this possibility.

We decided to use something in between these two extremes: We create one page describing how to perform a process step. As the wizard offers an expert and a novice mode, this page can be displayed to novice users and it can be hidden for experts, who do not need this information. In addition the atomic operations are spread over several pages. By default 5 atomic operations are grouped on one page. But someone who knows how the user interfaces of the automated atomic operations look like can group the operations in such a way that the automated operations fit well on the page. Therefore the "Additional Options" mode of WGT exists. This mode allows to group the atomic operations on the pages in a way that best fits the needs.

Figure 6 shows how a user can group atomic operations to pages. You can see that for each process step it is shown, which operations are needed in this step. On the right site it is possible to see which operations are placed on which page. By selecting an operation and pressing the arrows on the right you can move this operation to a page before or behind the actual page. Selecting for example the operation "If the size does not fit the requirements, change it" and pressing the "Down"-Arrow on the right would move this operation to the second page.



Replace graphical elements
Delete graphical elements
Add graphical elements
Performing changes according to style guides
Changing company naming
Rearrange text elements and graphical elements

Functions:	#
Detect all graphical elements	1
Decide for each element, if it is conform to the style guide	1
Find a new element for each non conform element	1
Check the size of each new element	1
If the size does not fit the requirements, change it	1
Replace all elements that have to be changed by the new ele...	2

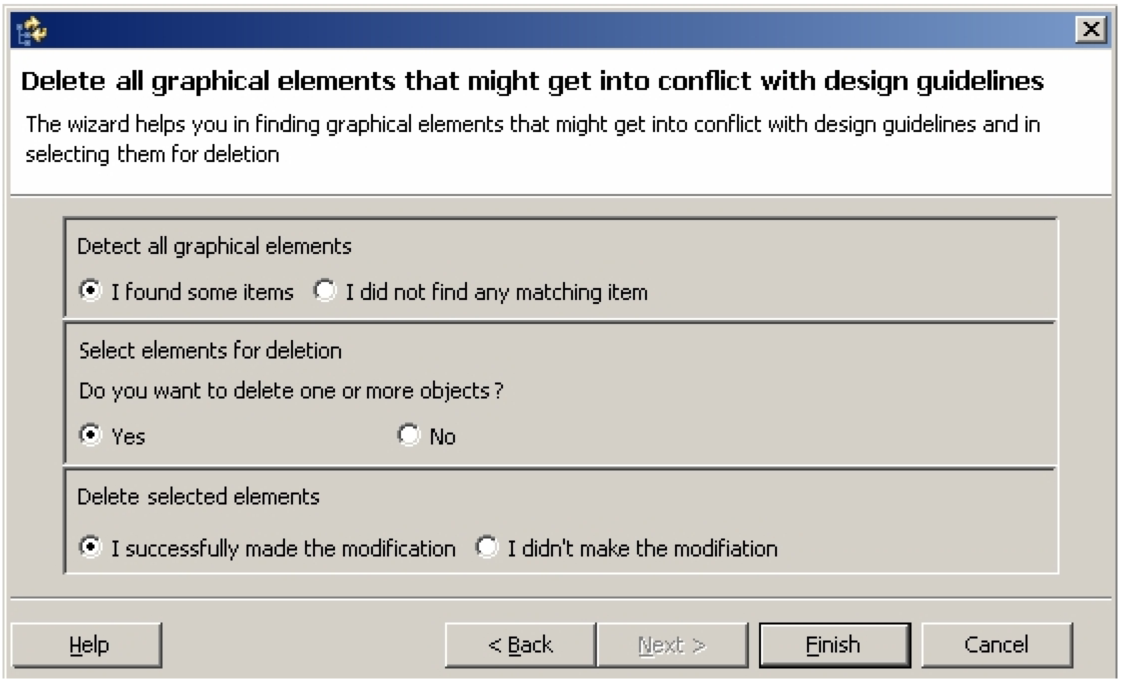
Figure 6: Arranging atomic operations on pages.

8 Step 3: Extension of Prototypic Wizard

Together with the process descriptions generated during the first step the wizard serves as a basis for communication between process experts and developers. Based on this prototype a common understanding of the process described in the wizard can be established. The wizard makes it easier to discuss ideas for further development in a vivid way. In addition it provides a code skeleton that can be enhanced comfortably by a developer. For this purpose the code generated by WGT is marked with special comments indicating, where additional code can be added. This can be done in the third step of the proceeding presented in this paper. The two steps presented so far are both executed by process experts. Step 3 has to be executed by a developer.

To make work faster and less error prone it is useful to add automated functionalities to the wizard, where this is reasonably possible. Therefore the wizard created by a process expert has to be handed over to a developer. The developer gets the information created in the first step and the prototype generated in the second step.

With the Wizard Generation Tool WGT for each process step one or more pages have been created. The pages contain a list of all operations needed to execute the process steps. The developer can add additional source code for each operation that can be automated. The initial source code contains comments indicating where automation is possible. The developer has to provide a new part of source code describing the user interface and a code section in the controller class that stores the information needed for and provided by the operation. The operations itself are stored in a so called function pool. This allows to reuse operations in several process steps. Comments within the original source code give hints, where to enter the new code and which dependencies between model, view, and controller have to be taken into account. By this it is possible to add code for all operations that can be automated. Figure 7 and figure 8 show a part of one wizard before and after adding automation to the operations. Together with the process expert the developer can check for each operation if it works in the desired way. By this proceeding we enhanced the prototype created based on the adaptations patterns to a fully functioning support tool for adaptation processes.


Delete all graphical elements that might get into conflict with design guidelines

The wizard helps you in finding graphical elements that might get into conflict with design guidelines and in selecting them for deletion

Detect all graphical elements

[✓ I found some items] I did not find any matching item

Select elements for deletion

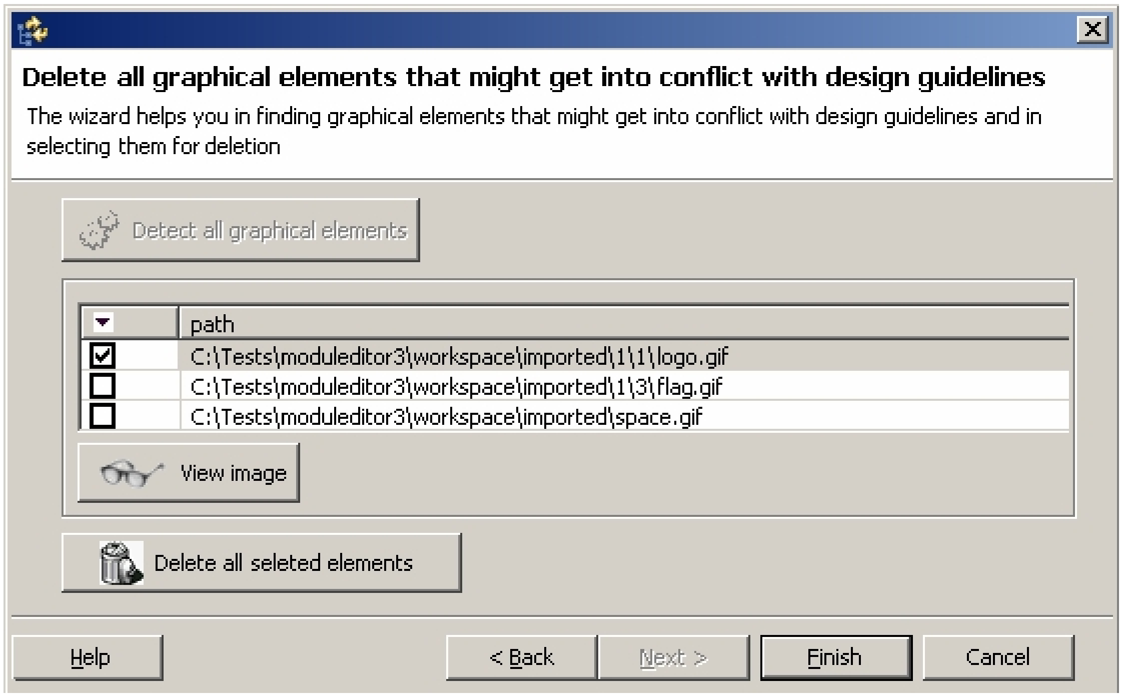
Do you want to delete one or more objects ?

[✓ Yes] No

Delete selected elements

[✓ I successfully made the modification] I didn't make the modifiation

Figure 7: One wizard page before adding automated functionalities.



Delete all graphical elements that might get into conflict with design guidelines

The wizard helps you in finding graphical elements that might get into conflict with design guidelines and in selecting them for deletion

	path
☑	C:\Tests\moduleditor3\workspace\imported\1\1\logo.gif
☐	C:\Tests\moduleditor3\workspace\imported\1\3\flag.gif
☐	C:\Tests\moduleditor3\workspace\imported\space.gif

Figure 8: The same page after adding automated functionalities.

9 Application of the Approach to the Example Scenario

With the approach presented before we created a wizard supporting users in performing the adaptation processes described in chapter 2. The wizard is based on patterns describing the following adaptations:

- Adaptation to a changed (corporate) design
- Adaptation in order to achieve a print version
- Adaptation to a changed terminology
- Adaptation to a changed language (i.e. translation)
- Adaptation to achieve an accessible version

For each adaptation a pattern exists describing the adaptation process. (The adaptation to achieve an accessible version is an exception: This adaptation is described by several patterns, used to describe sub-processes.) By entering the patterns into PIT and by adding all needed additional information about the processes, like process step descriptions and atomic operations descriptions, we created process descriptions. Then we generated a wizard with WGT based on these process descriptions. Outcome was a first prototype wizard (Figure 7 shows one page of this wizard).

As a next step we automated all parts of the wizard, where this was reasonably possible. (Figure 8 shows one of the pages enriched with automated functionalities.) We analysed the functions needed to perform the supported adaptation processes and we designed functionalities that could be used to enhance the prototype. These functionalities have been added to the wizard. Outcome of this proceeding was a tool that supports users in performing adaptation processes for E-Learning material.

A first evaluation with test users of the Content Sharing Project was promising: The users were enabled to perform all offered adaptations correctly. They found process guidance and detailed help on all processes. Based on their feedback we improved the functionalities. In a second, larger evaluation we tested our tool by comparing it to a common WYSIWYG HTML editor. At the moment no tool exists that supports the adaptation of existing E-Learning material to changed usage scenarios. Therefore we have chosen a tool that supports at least many functions needed to perform the adaptation processes to compare it with our tool. As many E-Learning courses are stored in HTML format we have chosen an HTML editor. We wanted a tool that is easy to use and that offers a WYSIWYG function allowing to control directly what has been changed. We decided to use Netscape Composer as HTML editor, as it is easy to use and allows to perform at least some typical adaptations.

We asked 32 users to perform some typical adaptations to three existing E-Learning courses. (It was possible to perform all adaptations with our tool as well as with Netscape Composer.) One E-Learning course was dealing with medical topics, one was an introduction to Multimedia, and the third one was a course to learn English. Half of the users were asked to work with our tool; the others got the WYSIWYG tool. Both groups got a detailed explanation how to use their tool. The tasks were the same for both groups. At the end of the test the participants were asked to answer a questionnaire in order to determine how satisfied they were in working with the tool.

Both groups were able to perform the adaptations as described in the manuals. Both groups did the tasks fast and with only very few errors. But we found that users working with our adaptation support tool needed less time to perform the tasks (in average 14 minutes with our tool compared to in average 20 minutes with Netscape Composer). And they made fewer mistakes. (Users working with the Netscape Composer made in average twice as many mistakes as users working with the adaptation tool.) In addition they were more satisfied with the use of our tool. As the adaptation tool got very positive feedback and the outcome of the adaptations had a very good quality, we think that our tool offers a better support for adaptation processes then the tool used for comparison. In addition we think that the knowledge collected with patterns is a good support for users in performing adaptations as the explanations how to use the tools for both test groups were based on the patterns. And the results for both test groups were very satisfying. For the future we plan to enhance our adaptation tool based on the feedback we got from the users.

All users, that have been taking part in the evaluation, have knowledge of the HTML file format. But one additional benefit of our tool is that this knowledge is not needed, as the tool abstracts from a concrete file format. Thus it is possible to use one tool to perform all adaptations in all files belonging to a learning resource without having detailed knowledge how of the formats. This is a feature that is not supported by the tools that up to now have been used to perform adaptation processes.

10 Evaluation of Concept

The aim of the concept presented in this paper was, to enable adaptation process experts without knowledge of process modelling to describe adaptation processes by an easy to understand process description formalism and to create prototypic wizards that reflect the processes as they are performed by the process experts.

To evaluate if process experts are able to use the process description formalism and the wizard generation tool independent from their knowledge of common programming formalisms and modelling formalisms, we performed a user test with 32 users. Half of the persons had knowledge of process modelling and of IT related issues like programming. Half of the users did not have this kind of knowledge.

All persons were asked to describe the same process with the help of PIT and to generate a prototype wizard based on the process descriptions by using WGT. It turned out that all users made very good process descriptions with only very few errors. Users without modelling and IT knowledge made a few more errors then users with this kind of knowledge. (In average the difference was only one error.) All users were able to generate a good prototype. It was no difference between the two user groups. In addition all users gave a very positive feedback regarding the understandability and the manageability of both tools. Thus we assume that all users were able to use the process description formalism and the wizard generation tool in the intended way. We also got some feedback how to increase the usefulness of the tools and the resulting wizard. We plan to analyse this feedback and to take it as a basis for further improvements.

11 Unresolved Issues and Future Work

There are several unresolved issues concerning PIT and WGT. For the future we plan to work on these issues. In this section we give an overview on the unresolved issues.

PIT allows entering relationships between processes. These are taken into account when generating the wizard: If one process has been finished in the wizard, a hint to related processes is given. PIT also allows specifying forces and consequences. But at the moment these are not taken into account in the generated wizard. But we are thinking on how to realize this. One possibility would be to use the forces as well as the consequences to find out if a process is useful in a specific situation:

If users are not sure which process they have to perform, they can search the problem statements. This helps to limit the number of possible processes. In addition users should be able to browse through the consequences and forces. This also reduces the number of possible processes. If a process has been performed the consequences can again be shown to users to allow them, to decide if a second process is necessary to eliminate negative consequences.

At the moment the wizard generation tool WGT is a first prototype. The layout of the wizard has to be overdone by taking into account common HCI guidelines. Nevertheless the wizard as it is by now already offers a good starting point for further development and a valuable basis for communication between process experts and software developers, as the evaluation of the adaptation tool, which is based on such a wizard, has shown.

Additionally to serving a basis for communication, the prototype wizard also can be used as a starting point for further development. Therefore WGT adds comments to the automatically generated source code of the prototype. Those comments offer hints to a developer, where it is possible to change the source code of the prototype in order to add additional functionalities. But it might occur that a process expert decides to change the process description after a while and to create a new prototype wizard based on the changed description. The new wizard then does not contain the enhancements added by the developer before. At the moment this means that the developer again has to add the additional functionalities. For the future it is planned to develop a concept that allows to merge both versions.

We created the approach presented in this paper to support adaptation processes. But we believe that it also can be used for other kinds of processes. Thus we tested for several other kinds of processes, if it is possible to describe them with PIT.

We found that PIT also can be used to enter already existing patterns (that might have to be adapted to the pattern notation used here) as well as new patterns. We have successfully tested this with some of the security patterns presented in [15]. In addition we created new patterns describing processes for hiring new employees as well as processes for booking journeys. For all these kinds of processes it turned out that it was possible to describe them with PIT and to create prototype wizards with WGT.

But it seems that there are other processes that cannot be described with PIT, e.g. processes with a focus on data flow. Thus, for the future it would be desirable to analyse which kinds of processes can be described with PIT and for which kinds of processes the approach presented here does not work.

When we were creating the E-Learning material adaptation wizard (compare section 9), we entered patterns as descriptions for the process and additional descriptions (not patterns) for the needed process steps and atomic operations. But as PIT only reassures that all needed information to perform a certain process is provided in the predefined structure, it has to be taken into account that the process descriptions provided by process experts might not meet common pattern criteria like being generic or having at least three known uses. It might occur that the process experts only create process descriptions that are written down in a pattern based notation formalism but do not meet common pattern criteria like having at least three known uses. But these descriptions also contain valuable knowledge and offer a good basis for the wizard creation. As the process expert is enabled to generate the wizard he or she can change the process description as many times as needed to achieve a wizard that really meets the experiences of the process experts.

12 Acknowledgements

The authors thank all persons, who supported this work. Especially we thank Michael Weiss, who provided a huge amount of helpful comments during shepherding for EuroPLoP 2008. Although many thanks go to our writer's workshop group at EuroPLoP 2008 for a very good discussion and so many useful hints.

The authors thank SAP AG - SAP Research CEC Darmstadt, as well as KOM Multimedia Communications Lab at the Technical University of Darmstadt for supporting this work.

This work is supported by the German Federal Ministry of Economics and Technology in the context of the project Content Sharing.

13 References

[1] Business Process Modeling Notation Specification. Final Adopted Specification, 2006. Available under: http://www.omg.org/docs/dtc/06-02-01.pdf

[2] Czarnecki, K., Eisenecker, U. W.: Generative Programming - Methods, Tools, and Applications. Addison-Wesley, 2000

[3] DIN En ISO 9241: Ergonomics of Human System Interaction. Part 11: Guidance on usability and part 110: Dialogue principles.

[4] Dreyfus, S.E.,Dreyfus, H.L: A five−stage model of the mental activities involved in directed skill acquisition. Unpublished report supported by the Air Force Office of Scientific Research (AFSC), USAF (Contract F49620−79−C−0063), University of Califonia at Berkley, 1980.

[5] Eckstein, R.: Java SE Application Design With MVC. 2007. http://java.sun.com/developer/technicalArticles/javase/mvc/index.html

[6] Folmer, E.,van Welie M., Bosch, J.: Bridging patterns: An approach to bridge gaps between SE and HCI. In: Information and Software Technology, 48(2), 2006.

[7] Fowler, M.: Analysis Patterns: Reusable Object Models. Addison-Wesley, 1996.

[8] Hahsler, M.: Analyse Patterns im Softwareentwicklungsprozeß. PhD thesis at WU Wien, 2001.

[9] Müller, S.: Modellbasierte IT-Unterstützung von wissensintensiven Prozessen - Dargestellt am Beispiel medizinischer Forschungsprozesse. PhD thesis at the Universität Erlangen - Nürnberg, 2007.

[10] PLML, XML DTD available under http://www.hcipatterns.org/PLML+1.0.html

[11] Robertson, S.: Requirements trawling: techniques for discovering requirements. In: International Journal of Human-Computer Studies, Volume 55, Number 4, October 2001.

[12] Royce, W., W.: Managing the Development of Large Software Systems. In: Proceedings of the 9th international conference on Software Engineering. 1987.

[13] Scheer, A.-W.: CIOs entwickeln sich zu Chief Process Officers. In: CIO, 2007. http://www.cio.de/karriere/cios_im_portrait/810380/index4.html

[14] Shneiderman, B., Plaisant, C.: Designing the user interface. Addison Wesley, 2004.

[15] Schumacher, M., Fernandez, E., Hybertson, D., Buschmann, F., Sommerlad, P.: Security Patterns - Integrating Security and Systems Engineering. John Wiley & Sons, 2005.

[16] Siau, K., Ericksson, J., Lee, L.: Theoretical versus practical complexity: The case of UML. In: Journal of Database Management 16, 3, 2005.

[17] Völter, M., Stahl, T.: Model-Driven Software Development: Technology, Engineering, Management. Wiley, 2006.

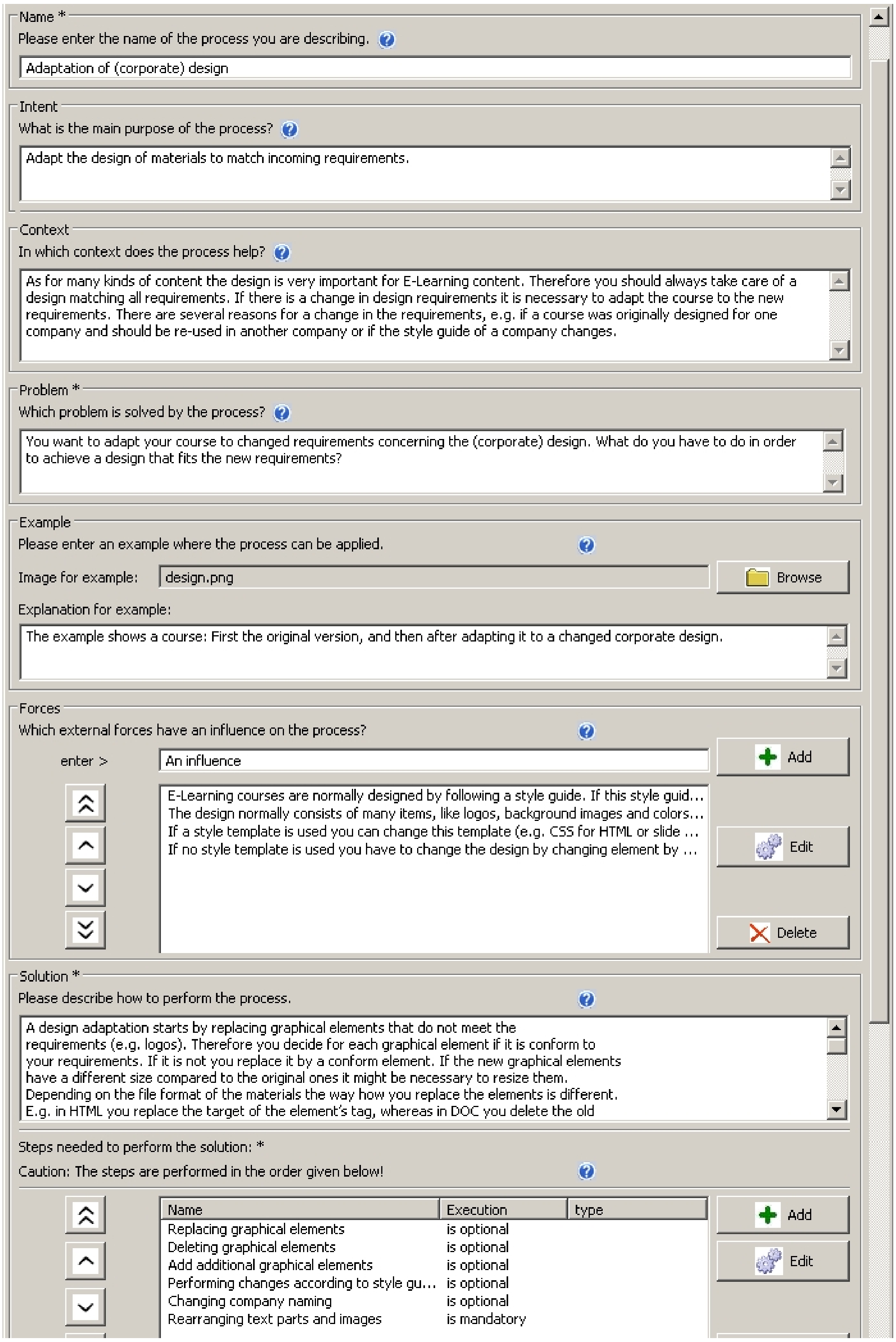
[18] Zimmermann, B., Bergsträßer, S., Rensing, C., Steinmetz, R. A Requirements Analysis of Adaptations of Re-Usable (E-Learning) Content. In: In Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2006.

[19] Zimmermann, B., Rensing, C., Steinmetz, R.: Patterns for Tailoring E-Learning Materials to Make them Suited for Changed Requirement. Published in the Proceedings of VikingPLoP 2006.

[20] Zimmermann, B., Rensing, C., Steinmetz, R.: Patterns towards Making Web Material Accessible. Published in the Proceedings of EuroPLoP 2007.

14 Appendix

The Process Description Input Tool PIT supports adaptation process experts in creating process description of adaptation processes. It has been described in section 6. Figure 9 shows the main input form of PIT.



Name *

Please enter the name of the process you are describing.

Adaptation of (corporate) design

Intent

What is the main purpose of the process?

Adapt the design of materials to match incoming requirements.

Context

In which context does the process help?

As for many kinds of content the design is very important for E-Learning content. Therefore you should always take care of a design matching all requirements. If there is a change in design requirements it is necessary to adapt the course to the new requirements. There are several reasons for a change in the requirements, e.g. if a course was originally designed for one company and should be re-used in another company or if the style guide of a company changes.

Problem *

Which problem is solved by the process?

You want to adapt your course to changed requirements concerning the (corporate) design. What do you have to do in order to achieve a design that fits the new requirements?

Example

Please enter an example where the process can be applied.

Image for example: design.png

Explanation for example:

The example shows a course: First the original version, and then after adapting it to a changed corporate design.

Forces

Which external forces have an influence on the process?

enter > An influence

E-Learning courses are normally designed by following a style guide. If this style guid...
The design normally consists of many items, like logos, background images and colors...
If a style template is used you can change this template (e.g. CSS for HTML or slide ...
If no style template is used you have to change the design by changing element by ...

Solution *

Please describe how to perform the process.

A design adaptation starts by replacing graphical elements that do not meet the requirements (e.g. logos). Therefore you decide for each graphical element if it is conform to your requirements. If it is not you replace it by a conform element. If the new graphical elements have a different size compared to the original ones it might be necessary to resize them.
Depending on the file format of the materials the way how you replace the elements is different.
E.g. in HTML you replace the target of the element's tag, whereas in DOC you delete the old

Steps needed to perform the solution: *

Caution: The steps are performed in the order given below!

Name	Execution	type
Replacing graphical elements	is optional	
Deleting graphical elements	is optional	
Add additional graphical elements	is optional	
Performing changes according to style gu...	is optional	
Changing company naming	is optional	
Rearranging text parts and images	is mandatory	

Figure 9: Part of PIT's input form.

PIT stores the process description given by the process experts in XML format (compare section 6). The following listing shows the DTD of the XML files containing the process information.

```
<!ELEMENT pattern (intent?, context?, problem, example_illustration?,
example_explanation?, forces, solution, process_steps, known_uses*,
consequences, related-patterns>
      <!ATTLIST pattern patternID ID #REQUIRED
                        confidence CDATA #IMPLIED
                        name CDATA #REQUIRED >
<!ELEMENT intent (#PCDATA)>
<!ELEMENT context (#PCDATA)>
<!ELEMENT problem (#PCDATA)>
<!ELEMENT example_illustration (#PCDATA)>
<!ELEMENT example_explanation (#PCDATA)>
<!ELEMENT forces (force*)>
<!ELEMENT force EMPTY>
      <!ATTLIST force name CDATA #REQUIRED>
<!ELEMENT solution (#PCDATA)>
<!ELEMENT Process_steps (Process_step+)>
<!ELEMENT Process_step EMPTY>
      <!ATTLIST Process_step
            name Name #REQUIRED
            mandatory (true | false) "true">
<!ELEMENT known_uses (#PCDATA)>
<!ELEMENT consequences
       (positive_consequence+, negative_consequence*)>
<!ELEMENT positive_consequence EMPTY>
      <!ATTLIST positive_consequence name CDATA #REQUIRED>
<!ELEMENT negative_consequence EMPTY>
      <!ATTLIST negative_consequence name CDATA #REQUIRED>
<!ELEMENT related_patterns (related_pattern*)>
<!ELEMENT Related_patterns (Related_pattern*)>
<!ELEMENT Related_pattern EMPTY>
      <!ATTLIST Related_pattern
            name CDATA #REQUIRED
            patternID ID #REQUIRED
            type CDATA #REQUIRED
      >
```

Listing 1: Pattern file DTD.

The information about the process flow is stored as a process graph. The following listing shows a part of such a process graph stored in XML (compare section 6.1). You can see a process identified via its ID. All process steps needed to perform the process are listed in the `requires` section of the process. For each process step it is noted if the step has to be performed (`mandatory="true"`) or not (`mandatory="false"`).

The last process step in the example is only performed if at least one of the steps before has been performed. Therefore all process steps, which can be performed before this step, are listed in the `precondition` section of the last process step. For the first process step you can see all atomic operations needed to perform this step. Again it is written down for each operation if the execution is mandatory. In addition you can see the three kinds of atomic operations (query, decision, and execution). The original file also contains a section defining the needed atomic operations. This section is not shown in the listing below.

```
<process id="process_pattern$56667" process-pattern="true">
  <requires fragmentRef="process_step$22357234" mandatory="false"/>
  <requires fragmentRef="process_step$25517184" mandatory="false"/>
  <requires fragmentRef="process_step$28757034" mandatory="false"/>
  <requires fragmentRef="process_step$36740146" mandatory="false"/>
  <requires fragmentRef="process_step$43532108" mandatory="false"/>
  <requires fragmentRef="process_step$50025522" mandatory="true">
    <precondition fragmentRef="process_step$22357234"/>
    <precondition fragmentRef="process_step$25517184"/>
    <precondition fragmentRef="process_step$28757034"/>
    <precondition fragmentRef="process_step$36740146"/>
    <precondition fragmentRef="process_step$43532108"/>
  </requires>
</process>
<process-step id="process_step$22357234">
  <requires functionRef="query$28693719" mandatory="true"/>
  <requires functionRef="decision$26294026" mandatory="true"/>
  <requires functionRef="decision$24376617" mandatory="false"/>
  <requires functionRef="query$23537464" mandatory="false"/>
  <requires functionRef="decision$32687500" mandatory="false"/>
  <requires functionRef="query$33442589" mandatory="false"/>
  <requires functionRef="execution$5472454" mandatory="false"/>
  <requires functionRef=" execution $51555041" mandatory="false"/>
</process-fragment>
```

Listing 2: Part of a process graph file.