Tool-Supported Customization of UML Class Diagrams for Learning Complex System Models

Imed Hammouda Olcay Guldogan [⊤] Kai Koskimies
Tarja Systä
Institute of Software Systems, Tampere University of Technology
P.O. Box 553, FIN-33101 Tampere, Finland
[⊤] Nokia Corporation
Tieteenkatu 1, 33720 Tampere, Finland
{imed, kk, tsysta}@cs.tut.fi, olcay.guldogan@nokia.com

Abstract

To employ an existing software library, its structure should be first learned and the required elements should be identified. This can be challenging if the library is large and only a specific part of it should be comprehended. In this paper, we study the problem of learning complex software libraries modeled in UML. It is argued that the learning process can be supported with a tool environment that allows the customization of the UML model according to the context of the learner, stepwise and dynamically chosen learning tasks, and focusing on a particular learning concern at a time. We show how such an environment can be achieved based on the concept of a pattern, using existing tool support. We demonstrate the idea with a part of Symbian platform architecture. The approach is evaluated in a case study where a pattern-driven learning environment is constructed for JPEG interchange file format specifications.

1 Introduction

Learning complex systems is a challenging task. The learning curve often depends on what experience the learner has and how familiar she is with the system domain. Beginners (for example students), in particular, may face serious problems in understanding the details of complex systems like various software platforms, standard API's or formats.

The understandability of a complex system can be improved by graphical representations. A good candidate for the graphical description of software-related systems is the Unified Modeling Language (UML)[20], which is a widely used standard design notation and thus assumingly familiar for most software engineers. Such a graphical description effaces unessential technical details like the intricacies of a particular textual syntax or implementations of methods, and shows the abstract relationships between the different parts of the system. In this paper we assume that the system description is given as a UML class diagram.

The problem is, however, that even the purely logical, essential information can be overwhelming for a particular person who is interested in applying (and understanding) the system only in her own, limited context. If we could somehow show this person only that part of the system description (e.g. a UML model) which is relevant for her purposes, the target of the learning process would be greatly reduced.

On the other hand, it is well-known that the comprehensibility of a system is mainly determined by the possibility of studying the system one part at a time [15]. Such a part can be a component, a subsystem, or even an aspect [1]; the important thing is that the learning task can be divided into pieces which make sense from the learner's viewpoint. This makes it possible for the learner to focus on one issue at a time, and to restrict herself to those issues that are relevant for her.

The set of potential learning issues can be anticipated beforehand, but the individual learner eventually decides which issues to focus on. However, since the selection of the relevant topics itself requires certain understanding of the system, we cannot assume that the learner is able to make deterministic choices concerning the learning issues in the beginning of the process. Instead, the learning process should be interactive, allowing the learner to select the learning paths dynamically as she understands the system better.

Separate learning issues typically contain sequentiality; certain pieces need to be learned before the other ones can be fully understood. For instance, to learn how to use a

certain feature provided by a software library, the engineer needs to understand how its individual parts relate to each other and in which order the individual implementation steps should be taken. Software documentation seldom provides that kind of support.

Thus, our research problem is the following: what kind of technique would support a learning process that (i) is based on the UML model of the target system, (ii) allows the customization of the UML model according to the needs of the learner, (iii) allows the learner to make dynamic choices concerning the subjects to be learned, and (iv) supports sequentiality in specific learning concerns.

We argue that these requirements can be satisfied technically using so-called patterns. In this context, a pattern is a specification of a configuration of modeling elements. Thus, a pattern is a generic concept that can be used to present the structure of virtually any collection of logically related (software or modeling) elements. We argue that a learning issue can be conveniently specified as a pattern on the UML model of the system. In the sequel we will use the term comprehension pattern for such patterns.

Separation of concerns (SoC) can be used to factor out different aspects (concerns) of a model [5]. We propose the use of SoC to decompose and organize the tasks required for learning a complex system into separate comprehensibility concerns. Such a concern covers all the model elements relevant for learning a certain part or aspect of the system. We will represent a comprehensibility concern as a set of comprehension patterns, where each comprehension pattern takes care of a more limited learning issue pertaining to the concept. We will show in this paper how an existing tool providing generic pattern support in UML environment [12] can be used to implement the idea of comprehension patterns and comprehensibility concerns in a way that satisfies our requirements (i) - (iv). Additional benefits for comprehension patterns offered by this tool include:

- A task-driven learning environment: The tool generates a task list on the basis of comprehension patterns, to be carried out by the learner. This kind of stepwise learning-by-doing paradigm makes the learning process more natural.
- Immediate feedback: When the learner makes a mistake, the tool immediately generates a new task for correcting the mistake. Usually, the learner is provided with tool-supported repair functionality.
- Informal explanations: Comprehension patterns can be augmented with informal explanations concerning the roles of the model elements in a pattern. Such explanations can be shown by the tool when the learner focuses her attention to a particular element.

In order to demonstrate our approach we use the MADE tool [12] to model a collection of comprehension patterns for the JPEG interchange file format specifications [18].

The remaining of the paper is organized as follows. In Section 2 we discuss comprehension patterns in more detail. Section 3 gives an introductory example of the application of our approach in the case of Symbian OS architecture. In Section 4 we present the tool platform we are using, and in Section 5, we briefly explain the JPEG interchange file format specifications and show how comprehension patterns and comprehensibility concerns can be identified and used in this domain. Related work is discussed in Section 6. Finally, we conclude with some remarks concerning future work in Section 7.

2 Comprehension Patterns

A pattern is an arrangement of software elements for solving a particular problem. Depending on the nature of the problem, we may speak of analysis patterns [9], architectural patterns [3], design patterns [3], coding patterns [3] etc. In the sequel we will give a simple structural characterization of a generic pattern concept.

To be able to define a pattern independently of any particular system, a pattern is defined in terms of element roles rather than concrete elements; a pattern is instantiated in a particular context by binding the roles to concrete elements. A role has a type, which determines the kind of software elements that can be bound to the role; the set of all the role types is called the domain of the pattern. Here we assume that the domain of a pattern is UML; that is, the roles are bound to UML model elements.

Each role may have a set of constraints. Constraints are structural conditions that must be satisfied by the model element bound to a role. For example, a constraint of association role P may require that the association bound to P must appear between the classes bound to certain other roles Q and R.

A cardinality is defined for each role. The cardinality of a role gives the lower and upper limits for the number of the instances of the role in the pattern. For example, if an operation role has cardinality '0..1', the operation is optional in the pattern, because the lower limit is 0.

In this work, we introduce a concrete pattern concept called a comprehension pattern whose purpose is to facilitate the learning of existing systems. Comprehension patterns are used to represent learning tasks in a form that makes it suitable for tool support. In the case of a comprehension pattern, binding a role to a model element is considered as a learning task. In this way it becomes clear to the learner which model elements play certain roles in the system. Assuming that each binding task is associated with informal explanations of the role to be bound (as is the case in our tool environment), the learner understands the pur-

pose of the related model element. Each binding task can be attached with a description detailing its objectives and the way it should be carried out.

We assume that each role in a pattern can be associated with the specification of a default model element, so that this model element can be generated and bound to the role automatically when the binding is requested, if the learner wishes. This is a key property in the tool, because it allows the generation of a customized version of the system model during the learning process, according to the learning paths the user has taken. Basically, the default model elements constitute roughly the complete system model. We will study the principles of pattern construction in the next section.

When modeling a comprehension pattern, we distinguish between two kinds of roles. Roles, which represent model elements existing in the original model (called original model roles) and roles that stand for possible model extension (called model extension roles). An original model role may have cardinality '1' or '0..1'. A '1' cardinality means that the model element represented by the role is required and must be present in the customized model. A '0..1' cardinality signify that the corresponding model element is optional. Thus, if the role is not relevant from the viewpoint of the learner, she may ignore the role, which will have the effect that all the other roles which depend on this role through their constraints will not appear in the task list. For example, assume that a particular association end in the UML model has multiplicity '0..1'. If the learner knows that in her context there will actually be no instances of the class involved, she will ignore the corresponding role, with the effect that the class will never appear in the generated customized system model. More importantly, other classes depending only on uninteresting classes will not appear in the model, either. Thus large portions of the model that are not interesting for the learner can be omitted in the produced customized model. A model extension role, on the other hand, may have cardinality '1', '0..1', '1..*', or '0..*'. Compared to original model roles, there can be more than one model element played by a model extension role. The reason is that the learner can provide different kinds of extensions for the same extension point.

In software engineering, separation of concerns (SoC) refers to the ability to identify those parts of software that are relevant to a particular concept, goal, task, or purpose. Concerns are the primary motivation for organizing and decomposing software into smaller, more manageable and comprehensible parts. In fact, SoC has been applied to different kinds of concerns ranging from business logic, performance, and security to other development-process concerns, such as comprehensibility, maintainability, and traceability. In this work, we are particularly interested in the comprehensibility concern. Enhancing system comprehensibility can facilitate the learning process in complex domains since these domains are usually composed of numerous parts and complex interactions between these parts.

3 Example: Learning the Symbian Architecture with Comprehension Patterns

Symbian [19] is an open standard operating system for data-enabled mobile devices such as PDAs and smart phones. The standard architecture comes with a large number of components providing support for a wide range of applications and functionality. Figure 1 shows a UML class diagram describing an overview of the Symbian communications and networking facilities, which is a core part of the Symbian platform architecture. Every class represents a component within the architecture. In overall, the architecture covers four different application types: message applications, web applications, connectivity applications, and comms applications. More detailed information about each application type can be found in [19].

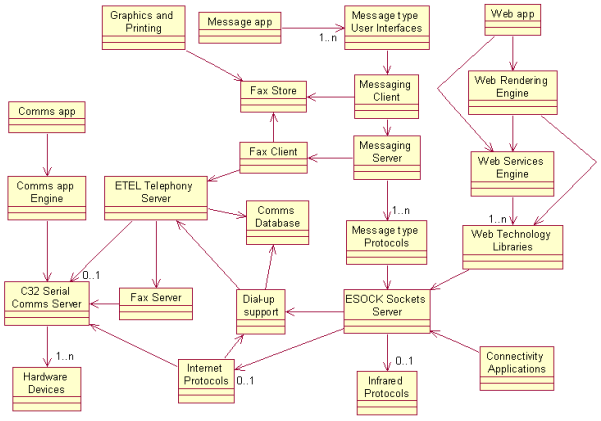

Figure 1. Original model

From the application type viewpoint, the architecture in Figure 1 can be decomposed into four main parts: each part represents the components that are relevant to each of the application types we have previously identified. Furthermore, it seems natural to define a separate comprehensibility concern for learning each of these four model parts. Therefore, we identify four comprehensibility concerns: messaging, connectivity, web, and comms (communications). The messaging comprehensibility concern, for example, covers all the model elements relevant for learning the messaging aspect of the platform.

The Symbian messaging comprehensibility concern can be further encapsulated by two comprehension patterns: The first pattern takes care of learning the elements that are shared by other application types such as the ETEL tele-

phony server, whereas the second pattern represents the elements specific to messaging only, such as ”Message type user interfaces”. The two patterns define roles for these model elements and constraints on their relationships. For example, there is a model extension role for extending the original model element ”Message app”. This role has cardinality ’1..*’ since the user can specify any number of own messaging applications types. The pattern defines another role for the ”Message type user interfaces”. In addition, the pattern defines a constraint on the uses relationship between the two roles meaning that the element bound to the model extension role uses the element bound to role ”Message type user interfaces”.

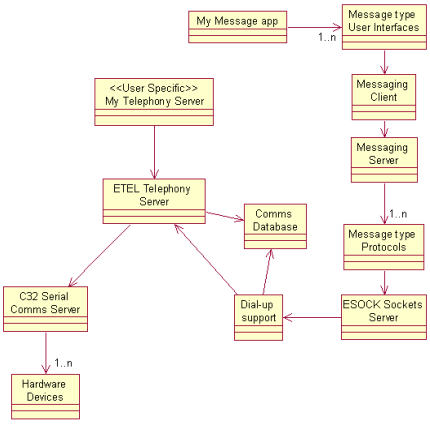

Figure 2. Customized model

When applying a comprehension pattern, binding a role to a model element is considered as a learning task. In our example, the user provides an element for the model extension role associated with the component ”Message app”. Based on the pattern specifications, a default name is suggested. The learner, however, uses her own element name ”My Message app”. The learner then acknowledges the task for providing an element for ”Message type user interfaces”. In this case, the default name specified by the pattern is used. When these two model elements are provided, a task for the uses relationship between the two elements is generated. Using the role constraints stored in the pattern, it can be checked whether the uses relationship appears between the right elements and that it is a one-to-many relationship. If the learner changes this relationship to one-to-one, a violation repair task, for restoring the relationship to one-to-many, is generated. In addition, the learner is given an optional task for creating and using the model element ”Infrared Protocols”, used for infrared connectivity support. The learner, however, might choose not to consider this component in her specific application so the task is not carried out and the corresponding element is not generated.

Figure 2 shows a customized UML model describing the components that should be considered when building a typical SMS messaging application. The developer needs to focus on the components that are only relevant to SMS messaging in general and on her application configuration in particular. Compared to the original platform model shown in Figure 1, the model in Figure 2 has undergone the following customization steps:

1. The concerns that are not relevant to messaging applications are dropped out. In our example, the elements that are only relevant to connectivity, web, and comms are left out.

2. Within the messaging concern, the elements that are not relevant to the specific application configuration are also dropped. The ”Internet Protocols” component, for example, is dropped since it is relevant to Email messaging but not to SMS messaging.

3. The optional model elements that are relevant to our specific application configuration are enforced. Even though the use of the ”C32 Serial Comms Server” (used for serial communications) is optional, the user thinks that every device should support this communication type. We, therefore, enforce its use in the customized model by changing the multiplicity of the corresponding association end from ’0..1’ to ’1’.

4. Using concrete names for the model elements that are application specific can enhance the understandability of the platform model. In the customized model, the messaging application component ”Message app” is replaced by a new component named ”My message app”.

5. The customized model can extend the original model by defining new elements that correspond to possible extensions in the platform. According to the Symbian specification, clients can provide custom extensions to the ”ETEL Telephony Server” component. This is illustrated in our example customized model by the new element ”My Telephony Server”.

4 Tool Platform: MADE

MADE [12] is an experimental platform for pattern-driven UML modeling. The platform itself is the result of the integration of a number of different existing tools. JavaFrames [10] and Rational Rose [16] represent the key components of the integrated environment. JavaFrames is a pattern-oriented task-based development environment built on top of the Eclipse [7] platform. The pattern concept implemented by JavaFrames is an instance of the generic pattern concept discussed previously. Rational Rose is a UML

modeling tool widely used by the industry. The communication between JavaFrames and Rational Rose is achieved through a UML model processing platform, xUMLi [2], providing a tool-independent API for accessing the UML models.

The MADE environment introduces patterns in the UML domain. MADE allows the specification of patterns and the binding of the roles of a pattern to UML model elements. The types of the roles are thus element types (metaclasses) of UML; for example, there are class roles, attribute roles, operation roles etc. The roles can be associated with various kinds of constraints on UML model elements. The constraints may refer to the elements bound to other roles, thus creating implicit dependencies between the roles. As an example, a naming constraint of a role may require that the name of the element bound to another role must appear in the name of the element bound to this role, according to a particular convention given as a regular expression.

The main contribution of the pattern tool [10] is that it maintains a dynamic task list for a partially bound pattern, indicating the binding tasks that are possible to carry out at a particular point of time. Recall that the action of binding a role to an (automatically generated) UML model element is here considered as a learning task. The tool keeps track of the mutual dependencies of the roles, reevaluates the constraints after each binding action, and updates the task list. Typically, the execution of a task generates new mandatory or optional tasks. The tool smoothly integrates patterns with UML models, so that the designer is free to edit the UML models as desired: if some of the constraints of a pattern bound to the model is broken as a result of an editing action, the tool generates immediately a new task to repair the broken constraint (and in some cases even an automated repairing option). The dynamic task list, integrated with the UML editor, allows the learner to proceed in a stepwise manner, understanding and producing the customized model one step at a time, and making editing experiments with the produced model.

The MADE pattern concept allows the specification of a partial order for the roles, implying that the binding of some roles must be carried out before certain other roles can be bound. This is used in our work for defining the order for individual learning tasks. The partial order is obtained simply by defining explicit dependency relationships between certain roles.

Another technically fairly simple, but in our context important contribution of the pattern tool is that one can associate informal instructions with each role, displayed when the binding task of the role is selected from the task list. In our work, we exploit this feature for informing the user about the meaning of the role (or the element bound to the role) from the viewpoint of the learning issue covered by the pattern. The tool also allows the highlighting of the model elements bound to the roles of a particular pattern, allowing the learner to easily see the essential parts of the learning issue. Finally, an essential property of the tool environment from the viewpoint of comprehension patterns is the possibility to associate a default UML model element for each role, to be automatically generated and bound to the role when the user performs a binding action (that is, a learning task).

5 Case Study: JPEG File Formats

In this section, we apply our approach to develop a learning environment for a software library representing the JPEG file format specifications.

5.1 Introduction to JPEG File Formats

Image and Video file format related software implementations have been widely used in practice. These systems are mostly complicated due to the underlying technical standards and specifications. Customizability and extendibility are desired characteristics for such systems, since they cover a broad range of considerations on the related field. In order to learn these systems, one has to learn their uses, considerations, and design models of the technical specifications. One of the well-known file formats for compressed images is JPEG (Joint Photographic Experts Group) [18]. JPEG is an image compression standard issued by ITU-T and JPEG committee of ISO in 1992.

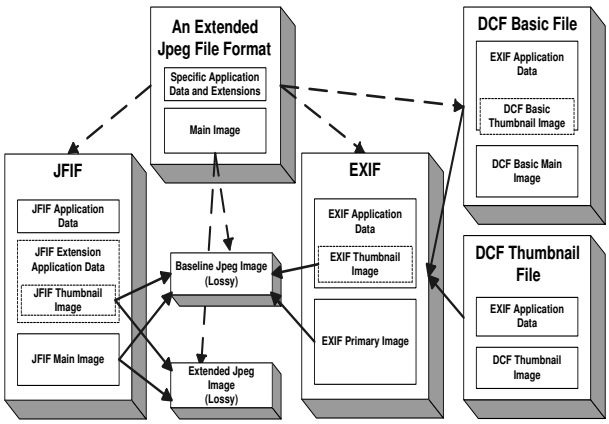

Figure 3. The JPEG file formats

JPEG compression processes cover various modes and methods for lossy and lossless compression in order to support a wide application scope. However, this wide coverage makes JPEG a complicated standard. Various file format specifications followed the JPEG standard to provide interoperability and simplicity of digital representation in practice.

JFIF (JPEG File Interchange Format) [11] is a relatively simple and widely used file format. EXIF (Exchangeable Image File Format) [13] and DCF (Design rule for Camera File system) [17] are mostly used by digital cameras for providing image-specific ancillary data. For simplicity and clarity of the examples in this paper, we refer to the EXIF file format containing only JPEG compressed data. Figure 3 illustrates the relationship between JPEG and related file formats.

5.2 Comprehensibility Concerns for JPEG File Formats

Based on the relationships presented in Figure 3, we have derived a comprehensibility concern architecture for JPEG file formats shown in Figure 4. Comprehensibility concerns are represented by circular shapes. Patterns are denoted by rectangular shapes whereas the arrows depict the inter-pattern dependencies. The dependencies show the order of the larger learning tasks. The following four concerns are illustrated:

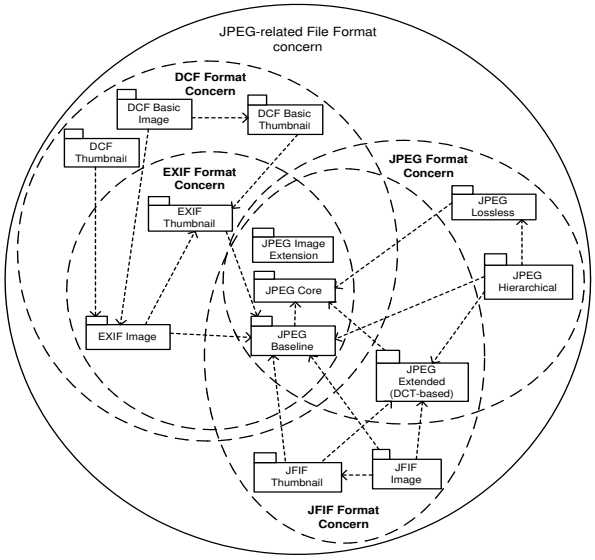

Figure 4. Comprehensibility concern architecture for JPEG-related file formats

- JPEG Format Concern represents JPEG digital image representation in the interchange format as defined in the technical specification. It is composed of six patterns: 'JPEG Core', 'JPEG Baseline', 'JPEG Extended', 'JPEG Lossless', 'JPEG Hierarchical', and 'JPEG Image Extension'. 'JPEG Core' represents the core structure common for all JPEG images. 'JPEG Baseline' pattern deals with the simplest JPEG file format, which is supported by every JPEG related application. 'JPEG Extended' pattern is responsible for representing additional lossy compression processes in the file format. 'JPEG Lossless' and 'JPEG Hierarchical' patterns are not significant for the example in this case. 'JPEG Image Extension' pattern defines extensions and/or restrictions for enhancing or adapting the file format for particular applications, such as a newer version of EXIF format.

- JFIF Format Concern abstracts the JFIF file format, which is based on JPEG file format. JFIF file format mainly defines basic restrictions and extensions on the JPEG file format. A JFIF image may include a thumbnail -a small copy of the original image- as well as the main image data. Thus, this concern is composed of two new patterns in addition to four JPEG patterns. 'JFIF Image' pattern handles the representation of the pre-defined restrictions and extensions on the underlying JPEG pattern. 'JFIF thumbnail' pattern is similar to the 'JFIF Image' pattern, however it doesn't define any extensions.

- EXIF Format Concern encapsulates the EXIF file format representation and related EXIF data structures. EXIF is a more complicated file format specification than JFIF. Similar to the JFIF concern, this concern contains two patterns for thumbnail ('EXIF Thumbnail') and main image data ('EXIF Image') in addition to three JPEG patterns. 'EXIF Image' pattern is mainly responsible for the primary image and associated ancillary data representation in the specified structure. 'EXIF Thumbnail' pattern handles the thumbnail image representation with the associated ancillary data. 'EXIF Image' pattern may use the 'EXIF Thumbnail' pattern, where the image contains thumbnail image.

- DCF Format concern contains three new patterns, which basically extend the EXIF-specific patterns, in addition to the EXIF patterns. 'DCF Basic Image' pattern is the significant pattern, which handles DCF main image and related restrictions and extensions. In contrast to JFIF and EXIF, DCF defines two types of thumbnail images. 'DCF Basic thumbnail' is similar to the aforementioned thumbnail images, and is contained inside the DCF image file format. 'DCF Thumbnail' is a stand-alone file format, which is independent from the main image.

5.3 EXIF Image Comprehension Pattern

Figure 5 shows a textual representation of the 'EXIF Image' pattern modeled in JavaFrames. For brevity, the figure describes a small part of the pattern structure. The left section displays a number of roles along with their constraints

Comprehension pattern: EXIF image	
Roles	Properties
ExifApplicationSegment : UML Class *(1)* **inheritance** : constraint **ExifIdentifier**: UML Attribute *(1)*	**description**: A JPEG Application Segment storing EXIF specific information. **taskTitle**: Add Application Segment to the EXIF image. **value**: ApplicationSegment **description:** constant identifier allowing the identification of an EXIF Application Segment.
ExifImage: UML class *(1)*	**description**: File format used for JPEG compressed images. **taskTitle**: Create the EXIF image.
contains: UMLAssociation *(1)* **participantA**: constraint **participantB**: constraint	**taskTitle:** Store Application Segment information into the EXIF image. **value:** EXIFApplicationSegment **value:** EXIFImage
GpsIfd: UML Class *(0..1)*	**description**: An optional IFD containing a set of tags for recording GPS information. **taskTitle**: Specify the Ifd strcuture for storing GPS information .

Figure 5. Representing 'EXIF Image' comprehension pattern in JavaFrames

and cardinalities whereas the right section entails other role properties. Role and constraint names are shown in bold font, role and constraint types are displayed in regular font, and cardinalities are specified in parentheses. The EXIFApplicationSegment role, for instance, represents a class role that stands for the application segment storing EXIF specific information. It has cardinality '1' since an EXIF image must have an EXIFApplicationSegment. The inheritance constraint associated with this role indicates that an EXIFApplicationSegment is an ApplicationSegment.

In addition, every EXIFApplicationSegment should have a constant identifier allowing the identification of an EXIF application segment. This is encapsulated by the UML attribute role EXIFIdentifier. The UML Association role ('contains') represents the composition relationship between the elements EXIFApplicationSegment and EXIFImage since an EXIF image stores an EXIF application segment. The relationship is further enforced by constraining its participants. This is shown in Figure 5 by the two constraints participantA and participantB. Furthermore, the figure shows a UML class role 'GpsIfd' with cardinality '0..1'. Such cardinality means that the element is optional. In our example, an EXIF application segment might or might not contain an IFD (Image File Directory) for recording GPS information.

As we have mentioned earlier, JavaFrames transforms the pattern specification into a task list. Figure 6 shows two learning tasks. Every task is associated with a task title and a task description as specified in Figure 5. A red dot (filled) marks a mandatory tasks whereas a white dot denotes an optional task. The upper task stands for providing an association for the UML association role specified in Figure 5. The other task represents the 'GpsIfd' role in Figure 5. This task is optional since the cardinality of the role it represents is '0..1'. In addition to displaying the tasks to be performed, the environment shows the learner how many mandatory and optional tasks are left to complete the actual customization session. This is shown in the title section of the figure.

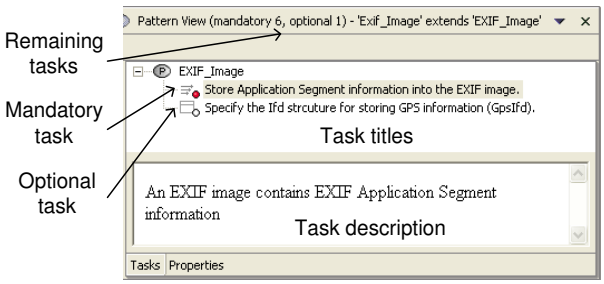

Figure 6. 'EXIF Image' learning tasks

5.4 An Example Customization Session

Figure 7 shows an example customizing session for learning the JPEG interchange file format specifications discussed in this paper. The patterns which have been identified in section 5.2 are shown in the bottom left view of the Figure. In the same view, the environment allows the user to choose between different JPEG concerns. Each concern corresponds to a specific set of features. The user, for example, might choose to learn the features of the EXIF image file format with or without the thumbnail information.

The learning tasks and their detailed description are shown in the bottom right view of the figure. Each task corresponds to understanding the role of a specific element in the standard. Figure 7 shows one mandatory task and two optional tasks. When a task is carried out (by double clicking on the task title), the corresponding model element is added to the system model and new tasks might be generated. The tool keeps track of all task information. This is important when the user wants to retrieve the description of the elements that have been constructed earlier. The recorded tasks are shown in the bottom middle view of the environment. Finally, the upper part of Figure 7 shows the customized model.

In a typical scenario, the learner chooses the part of the model she wants to customize/learn by selecting the corresponding pattern from the 'Concerns/Patterns' view. The learner then reads the available task descriptions and decides which learning step to take (in case more than one task is displayed). In the case of Figure 7, she might choose to provide the required model element 'IfdEXIF' by double clicking on the upper most task title. As a result, a new

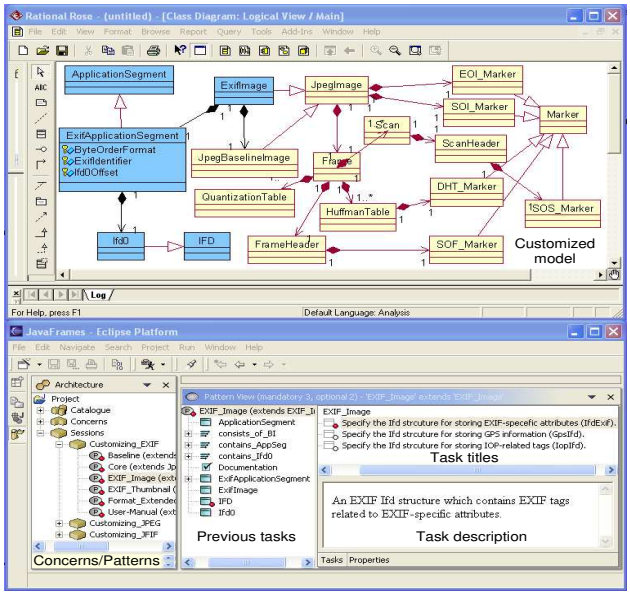

Figure 7. An example customization session

UML class named 'IfdEXIF' is added to the Rose view. The learner might proceed by generating more model elements. If the learner decides to quit the learning session, she has to save the generated model. She can later proceed with the session since all session information is automatically saved and recovered by the environment.

Using the tool, it is possible to highlight the elements that correspond to a specific pattern or concern. Figure 7, for example, shows in blue (darker) color the elements corresponding to the EXIF image information as specified earlier by the 'EXIF Image' pattern in the EXIF concern. Using this feature, the logical structure of the customized model can be retrieved at any phase during the customization session. This might enhance model comprehension.

5.5 Experiences

Figure 8 shows a comparison between the original model of the JPEG library structure versus several customized models with respect to the model elements that have been considered in the case study. The customized models represent different concerns in the library. Depending on the selected concern, the customization resulted in a reduced number of model elements. For example, this number is reduced by 50 % when the user focuses on the JFIF related features.

In order to evaluate the JPEG learning environment presented in this work, we have requested the feedback of two experts in the field of image processing (but not familiar with JPEG file formats structure): The first is a member of an image processing group at Tampere University of Technology and the second works for a telecom company. The questions that both learners had to answer fall into three main categories: the positive qualities, the negative aspects and the missing features of the environment. The main positive experiences were:

UML Model	Classes	Associations
Original Model	**70**	**50**
JPEG Baseline	**27**	**19**
EXIF without thumbnail	**49**	**37**
EXIF with thumbnail	**55**	**43**
Whole JFIF	**32**	**24**

Figure 8. The original model versus several customized models

- Using our approach, it was easier to understand and memorize the elements and the various relationships in the model. The "learning by doing" approach, in particular, seemed useful and contributed to model comprehensibility.
- From an industrial point of view, our methodology can be useful for companies that are interested in the same software components from different technical perspectives.

The main critics of the methodology were:

- Learning tasks should be carefully explained and the descriptions should be detailed.
- The order of the learning tasks should be carefully planned since there can be different paths when constructing models, for example the top down versus the bottom up approaches.
- Clustering the model elements that belong to the same concern into one abstract element can be a further enhancement of the approach. It should be then possible to expand this element to view the inner details.

Identifying comprehensibility concerns, and thus comprehension patterns, for learning a software system depends heavily on its architecture. If the architecture clearly separates different concerns, the comprehension patterns may at least partly follow those concerns. In this case the comprehension patterns are easier to find, but on the other hand there is less need for comprehension support. Comprehension patterns are especially valuable for systems with crosscutting concerns that are not directly reflected by the architecture. Once identified, it is a relatively easy task to model the patterns in MADE. For this, the roles and the properties

each pattern need to be specified. For example, it took a couple of hours to model the 'EXIF Image' pattern shown in Figure 5. This includes specifying the informal task descriptions shown to the learner.

6 Discussion and Related Work

6.1 Program Comprehension

Software comprehension can be seen as a task of building mental models, i.e., internal working representations of the subject software, of the underlying software at various levels of abstraction. In addition to static entities (such as text structures or plans), the mental models also contain dynamic entities: a dynamic entity might be a sequence of actions while following a plan to reach a particular goal [21]. Activities and approaches supporting software comprehension include software visualization, documentation generation, and reverse engineering. The goal of reverse engineering is to construct abstractions and design information from the subject system [4]. Tools supporting software visualization and reverse engineering often visualize the subject system using a graphical notation but rarely use the same notation as used for software design, namely, UML in the case of object-oriented systems. In addition, the importance of the dynamic entities of cognitive models is often overlooked in software comprehension tools. In the approach presented in this paper, the learner follows a step-by-step process that helps her in learning a specific learning issue.

Von Mayrhauer and Vans [21] point out a common inadequacy in several cognition models supporting program comprehension they compared: "Most models assume that the objective is to understand all the code rather than a particular purpose. While these general models can foster a complete understanding of a piece of code, they may not always apply to specialized tasks that more efficiently employ strategies geared toward partial understanding". In our approach, partial understanding is supported by dividing the model to be learned into learning concerns and further into comprehension patterns.

Erdem et al. propose a method and tool called MediaDoc to generate tailored software explanations [8]. Task and user models are first constructed by studying a series of user questions while performing a certain task. These models are then used in various ways to generate explanations tailored to user's task and expertise. The user can pose queries either by directly entering a query or by interacting with MediaDoc. MediaDoc either generates an explanation that directly answers the query, or reformulates the query as a new subtask. One of the motivating observations when developing MediaDoc was the importance of dialogues with experts when learning a system, even if a good documentation is available. One reason for this is that the dialogue between the learner and the expert facilitates the refinement of the questions [23]. In our approach, gathering the set of potential learning concerns is assumed. This could be done, e.g., by studying a series of user questions as done in MediaDoc. The clarifying learning issues, e.g., issues that need to be first understood, and the sequentiality of the learning process is captured using comprehension patterns. The expertise of the learner is not taken into account. Instead, the system models are customized based on the learning task only. Also, we do not allow the user to pose free-form textual questions, but rather aim at structuring the learning concerns and tasks in a manageable and well-defined way and mapping the directly with them system models under examination.

6.2 Learning Complex Structures

The approach presented in this paper represents an active learning environment for learning complex domain models. Active learning has been defined as the process of continuous and active construction and reconstruction of experiences [6]. The comprehension patterns in our methodology encapsulate the learning experience in the form of a set of learning tasks. The comprehension patterns themselves can be updated when more knowledge about the domain is obtained. Learning is then carried out by constructing the personalized system model as instructed by the patterns.

Constructivist learning systems have been discussed in [22]. The authors define the constructivist view of learning as a process in which learners play active roles in constructing the set of conceptual structures that constitute their own knowledge base. In our approach, the learner constructs her own conceptual structure of the system by creating personalized models of the complex domain. The learner, in addition, takes an active role in defining the learning environment. This is exhibited by the fact that the environment adapts the task descriptions to the concrete names given by the learner in earlier tasks. Moreover, depending on the chosen alternative, the environment behaves differently when the learner is presented with a number of alternatives.

From the viewpoint of structuring learning material, a similar approach to our work is presented in [14]. The authors propose an architecture for a goal-oriented way of teaching. They illustrate their concepts by presenting a web-based adaptive learning environment. The methodology described in [14] structures the domain knowledge of courses into three layers: the educational material, the concepts, and the knowledge goals. Similarly, we have structured the domain knowledge of complex system models into three layers: the model itself, the concerns abstracting the model, and the patterns encapsulating the learning tasks. Using the terminology of [14], our models represent the material, our concerns represent the concepts structuring this

material, and our comprehension patterns stand for the concrete learning goals.

7 Conclusions

In this work, we have described an approach for facilitating learning software libraries through the customization of the model entailing their structures. Our approach is to encapsulate the model into separate concerns. Every concern corresponds to a set of related features (matters of interest) in the software library. We further used comprehension pattern to encapsulate the smaller learning tasks within each of these concerns. We have applied our methodology to construct an interactive environment for learning a software library representing the JPEG file format specifications. The structural model of the JPEG library is composed of numerous components, interactions, and restrictions making it harder to learn and use. In order to model the JPEG library comprehension patterns, we have used a generic pattern-based development environment known as MADE.

Our early experiences suggest that our methodology contributes to the comprehensibility of system models. Stepwise learning by doing, separation of unrelated model features, grouping of related model elements, and customizing original models constituted the key benefits of our approach. Nevertheless, in order for our environment to be useful, the order of the learning tasks should be carefully planned and the descriptions of the tasks should be described in detail.

A possible direction for future work is to further evaluate the effectiveness of the presented methodology and tool support. In this regard, we might consider using the discussed case study to teach engineering students the fundamentals of the JPEG file format specifications. This assignment can be given in the scope of a Digital Image Processing course. We also consider applying our methodology to a more complex industrial case study.

Acknowledgments. This work is funded by the National Technology Agency of Finland (Tekes) and the Academy of Finland. The authors would like to thank Esin Guldogan and Mejdi Trimeche for their feedback on the case study.

References

[1] Communications of the ACM. Special issue on Aspect-Oriented Programming, 44:10, 2001.

[2] J. Airaksinen, K. Koskimies, J. Koskinen, J. Peltonen, P. Selonen, M. Siikarla, and T. Systä. xUMLi, towards a tool-independent UML processing platform. In *Proc. NWPER*, 2002.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley, 1996.

[4] E. Chikofsky and J. Cross. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17, 1990.

[5] T. Clark, A. Evans, and S. Kent. Aspect-oriented metamodelling. *The Computer Journal*, 46(5):566–577, 2003.

[6] J. Dewey. *Experience and education*. New York: MacMillon, 1938.

[7] Eclipse WWW site. Available at http://www.eclipse.org.

[8] A. Erdem, L. Johnson, and S. Marsella. Task oriented software understanding. In *Proceedings of ASE 98*, pages 230–239, October 1998.

[9] M. Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, Reading MA, 1997.

[10] M. Hakala, J. Hautamäki, K. Koskimies, J. Paakki, A. Viljamaa, and J. Viljamaa. Generating application development environments for Java frameworks. In *Proceedings of GCSE'01*, pages 163–176, Erfurt, Germany, September 2001.

[11] E. Hamilton. Jpeg file interchange format version 1.02, September 1992.

[12] I. Hammouda, M. Pussinen, M. Katara, and T. Mikkonen. UML-based approach for documenting and specializing frameworks using patterns and concern architectures. In the 4th workshop of AOSD Modeling with UML, October 2003. San Francisco, California.

[13] S. of Japan Electronics and I. T. I. Association. Exchangeable image file format for digital still cameras: Exif version 2.2, April 2002.

[14] K. Papanikolaou, G. D. Magoulas, and M. Grigoriadou. A connectionist approach for supporting personalized learning in a web-based learning environment. In *Proceedings of AH 2000*, pages 189–201, Trento, Italy, August 2000.

[15] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.

[16] Rational Rose WWW site. Available at http://www.rational.com/products/rose/index.jsp.

[17] J. E. I. D. A. Standard. Design rule for camera file system version 1.0, December 1998.

[18] C. R. T.81. Information technology - digital compression and coding of continuous-tone still images, September 1992.

[19] M. Tasker, J. Allen, J. Dixon, M. Shackman, R. T., and J. Forrest. *Professional Symbian Programming: Mobile Solutions on the EPOC Platform*. Wrox Press Inc, 2000.

[20] Unified Modeling Language WWW site. Available at http://www.uml.org/.

[21] A. Von Mayrhauser and A. M. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(2):44–55, 1995.

[22] L. Weiqi. Constructivist learning systems: a new paradigm. In *Proceedings of ICALT 2001*, pages 433–434, Madison, USA, August 2001.

[23] P. Wright. *Handbook of Human-Computer Interaction, Issues of content and presentation in document design*, chapter 28, pages 629–652. Elsevier Science Publishers B.V. (North Holland), 1988.