A LIGHTWEIGHT PROCESS MODEL AND DEVELOPMENT METHODOLOGY FOR COMPONENT FRAMEWORKS

Ansgar Scherp, Susanne Boll

OFFIS, Escherweg 2, D-26121 Oldenburg, Germany

ABSTRACT

Software frameworks typically constitute a semi-finished software architecture for a complex application domain that can be adapted to the needs and requirements of a concrete application in the domain. Since the introduction of object-oriented frameworks in the late eighties, the development of software frameworks is still costly and difficult to handle. To reduce development risk, process models and development methodologies for object-oriented frameworks have been developed, e. g., the hot-spot-driven approach by Pree. With the emergence of component technology also so-called component frameworks appeared that are currently considered as the highest level of software architecture reuse. In contrast to object-oriented frameworks, a proper process model for component frameworks is still missing. Such a process model needs support by a development methodology for identifying and specifying the framework's components. Consequently, the quality of today's component frameworks is eventually dependent on the experience and skills of the framework developers.

The contribution of this position paper is a lightweight process model and development methodology for component frameworks. This process model is called ProMoCF and bases on the hot-spot-driven approach for object-oriented frameworks. ProMoCF extends it by activities and methodical support for identifying the framework's components and defining the framework's flexibility requirements by introducing so-called group-hot-spot-cards. In this position paper, we argue for the benefits of a lightweight process model for developing software frameworks. The proposed approach has been successfully applied to the development of our component framework for personalized multimedia applications.

1. INTRODUCTION

The motivation for developing a software framework is often based on very good expertise in a specific domain and the possibility to sell several similar software products sharing common code. The knowledge application developers have gained by developing applications in a specific domain for years is abstracted into a software framework to ease reuse and to define a common architecture for the domain. A high-quality software framework is typically the result of a long development process. Hence, software frameworks should be developed by using a proper process model supported by a corresponding development methodology [1]. The process model describes which activities for developing the framework have when to be conducted. The corresponding development methodology supports the framework developers in how to conduct these activities. Process models with corresponding development methodology already exist for object-oriented frameworks, e. g., the hot-spot-driven approach [2]. However, an appropriate process model and development methodology for *component* frameworks is still missing.

In this position paper, we discuss three different types of process models: heavyweight, lightweight, and agile models. We argue for the benefits of a lightweight process model for developing software frameworks. Consequently, we present a lightweight process model and development methodology for component frameworks, which is called ProMoCF (short for "Process Model for Component Frameworks"). The basis of ProMoCF forms Pree's hot-spot-driven approach. It is extended by activities and methodical support for identifying the framework's components and specifying the framework's variability by group-hot-spot-cards.

Since it might not be obvious why and how a process model for object-oriented frameworks can be evolved towards component frameworks, we first clarify the concept of software frameworks and the differences between object-oriented and component frameworks in Section 2. Some general issues in regard of developing software frameworks are considered in Section 3. In Section 4, we argue for the benefits of a lightweight process model for developing software frameworks. Our lightweight process model and development methodology for component frameworks ProMoCF is presented in detail in Section 5. It has been successfully applied to the development of our component framework MM4U (short for "Multimedia for you") for personalized multimedia applications, which is described in Section 6, before we come to our conclusion.

2. SOFTWARE FRAMEWORKS

A software framework is typically a semi-finished software architecture for a complex application domain [3] that can be adapted to the needs and requirements of a concrete application in the domain. Historically grown, there exist two kinds of software frameworks, the object-oriented frameworks and

the component frameworks. They differ in the technology used for adaptation to the requirements of a concrete application: In case of object-oriented frameworks, the adaptation and specialization for a concrete application is conducted by overriding abstract classes or by different implementations of predefined interfaces. With component frameworks, specialization only takes place by composing different software components that implement the component framework's interfaces. In addition, also hybrid frameworks exist, which use both technologies for adaptation.

Object-oriented frameworks: According to Bosch [4] the probably most cited definition of object-oriented frameworks is the one by Johnson and Foote in [5] describing an object-oriented framework as a set of classes defining an abstract solution for a family of similar problems. This set comprises concrete and – in particular – abstract classes [6] providing an incomplete design and implementation for applications in a specific domain [7, 2]. So, an object-oriented framework includes the reuse of both design and code [4]. Many authors follow this definition, e. g., [3, 8, 2]. The behaviour of an object-oriented framework can be adapted to the requirements of a concrete application in two different ways [5]: The first way is by inheriting concrete classes from the abstract framework classes. This is called white-box usage of the framework. The second way for adapting the framework is by different compositions of concrete classes that implement the framework's interfaces. Thus, changing the framework's behavior is realized by different compositions of concrete classes and not by inheritance. This is called black-box usage of the framework. By partially opening a black-box-framework towards white-box usage "arbitrary shades of gray" can be build [3]. Typically, white-box frameworks mature to a black-box one when often reused [9, 5].

In the last couple of years, one can observe that often applications exploit more than one object-oriented framework to realize their functionality [7]. Using multiple, independently developed object-oriented frameworks for the construction of a single application often leads to crucial framework integration problems (see, e. g., [4, 10]). The interaction between different object-oriented frameworks can only be managed on an higher level that describes which parts of the object-oriented frameworks overlap or interact [3]. The idea here is to develop so-called *frameworks of second order*. In the way an object-oriented framework uses abstract classes and interfaces for combining sets of concrete classes, frameworks of second order could be used to combine abstract subsystems that are realized by frameworks of first order. Component frameworks are a step towards this goal.

Component frameworks: In [11], component frameworks are defined as collection of different components with a predefined cooperation behavior and the aim to fulfill tasks in a specific domain. Szyperski states in [3] more precisely that a component framework is a collection of rules and interfaces, which control the interaction of components within a component framework (q. v. [12, 13]). A component framework allows instances of components that implement the framework interfaces to be plugged in and executed by the framework. The component framework controls the interaction between the instances of the components [12]. In contrast to object-oriented frameworks, adapting the component framework's behavior to the requirements of a concrete application is only allowed by composition. However, instead of single concrete classes, different instances of the framework components (and their classes) are composed.

Having defined the concepts of object-oriented and component frameworks, we can now discuss some general issues in regard of developing software frameworks.

3. DEVELOPING SOFTWARE FRAMEWORKS

The development of software frameworks is characterized by an iterative and incremental development process, due to the complexity of frameworks and the frequent changes of the framework's flexibility requirements [4]. Frameworks are developed in many small iterations. The architecture and interfaces of the framework have to be redesigned again and again. Beginning with some example applications in the considered domain, relevant functionality is gradually abstracted and integrated into the framework. During the analysis of the application domain, it has to kept in mind that the behavior and the configuration of the framework can change for different concrete applications. On account of this, a process model for framework development should provide an explicit activity for identifying and specifying the framework's variation points, i. e., those parts of the framework, where application-specific functionality can later be integrated (cp. [14]). This activity should be supported by an appropriate framework development methodology. In the following, we briefly consider today's support for developing object-oriented frameworks and component frameworks.

Object-oriented frameworks: For developing object-oriented frameworks, a couple of process models and development methodologies can be found. The probably most known and most commonly used approach is the hot-spot-driven process model by Wolfgang Pree, which has been published in several papers and books, e. g., [15, 2, 16, 17]. The hot-spot-driven approach is a proved process model with corresponding development methodology for developing object-oriented frameworks. It uses hot-spot-cards to identify and manage the flexibility requirements of the object-oriented framework. Hot-spot-cards are a variant of the well known CRC-cards (Class Responsibility Collaboration cards) by Beck and Cunningham [18]. Other approaches for developing object-oriented frameworks are, e. g., the framework design by systematic generalization [19], a role-based approach for developing object-oriented frameworks [20], and a pro-

cess model with a comprehensive guideline for framework development [21].

Component frameworks: In contrast to object-oriented frameworks, a proper process model with methodical support for component frameworks is still missing. Consequently, the quality of component frameworks is eventually dependent on the experience and skills of the framework developers. To improve the development of component frameworks, the design of software components could be aligned in such a way that flexible and adaptable components are created. This means that software components are to be developed such that they can be easily used to assemble component frameworks. By implementing such flexible components, one hopes that this automatically leads to well designed component frameworks. However, these components are developed without considering the requirements to a concrete component framework in a specific domain. Thus there is the peril of adding more and more flexibility to the components that is not appropriate to the framework domain. However, it is important to deliberately implement flexibility into a component framework reflecting the requirements of the considered domain in a reasonable manner. Actually, adding unnecessary flexibility noticeably increases the complexity of a framework [15]. Consequently, the flexibility requirements to a component framework should be identified and specified by means of a proper process model and development methodology.

4. THE NEED FOR A LIGHTWEIGHT PROCESS MODEL FOR DEVELOPING FRAMEWORKS

The development of software frameworks is typically conducted by an iterative and incremental process model. However, there are different kinds of process models, like heavyweight, lightweight, and agile models. We argue that lightweight process models are most suitable for the development of frameworks:

Agile process models, e. g., the very popular eXtreme Programming (XP) [22] or Cockburn's Crystal Family [23], do not provide a real system specification. For example, in the case of XP only rudimental use cases, so-called *stories*, are defined to describe the application's functionality. This functionality, however, is only incrementally added to the application's design and only "the simplest thing that could possibly work" is implemented. Further flexibility requirements and the aspect of reuse of design are not considered. It is assumed that it is easier to change the original source code than adding flexibility in advance to provide support for reuse. This means that only as much flexibility is added to the application's design as it is needed to realize the actually required functionality. In other words, agile models do not target at developing reusable software architectures in the sense of the reuse characteristics of software frameworks. Although, agile process models like to exploit existing components to actually realize the required functionality and by this provide reuse of existing components, this does not help for developing reusable architectures, like component frameworks. As a consequence, we conclude that agile process models should not be used for developing frameworks.

Heavyweight process models, like the Rational Unified Process [24] and the Domain Specific Software Architecture (DSSA) engineering process [25], provide a fully-fledged support for developing reusable software applications (in a specific domain). When adapted to the requirements of a concrete domain, they provide an efficient development of applications in that domain. However, the management for evolving frameworks is more complex than the management of normal applications. Managing a software framework comprises not only the development and improvement of the actual framework but also the management of using the framework in possibly many concrete applications. Consequently, using a heavyweight process model for evolving frameworks generates an unnecessary management overhead and an increase of development costs.

Lightweight process models typically only define a few artifacts and require less management effort. When evolving a framework, its design and implementation is in constant change. All artifacts describing the framework and its functionality must be revised again and again. Consequently, it is not reasonable to write all artifacts documenting the framework in parallel to its development. To reduce management effort, only as much documentation should be written, as it is required to complete the documentation when finishing a framework development milestone, e. g., an official release of the framework for usage by external organizations (cp. [1]). Consequently, using a lightweight process model provides for an efficient management of evolving software frameworks at less cost. In the following section, we present ProMoCF, a lightweight process model for developing component frameworks.

5. MODIFICATION OF PREE'S APPROACH TOWARDS COMPONENT FRAMEWORKS

The motivation for modifying the hot-spot-driven approach towards a process model and development methodology for component frameworks is twofold:

- With the original hot-spot-driven approach aCiRC-cards (abstract Class/interface Responsibility Collaboration cards) are used to identify the abstract classes of the framework's object model. Also framework clusters are defined on the basis of key abstractions (see [15]). Both could be useful to identify the framework's components, however after defining them, they are unconsidered during the remainder of the process. To provide support for developing component frameworks, an explicit activity for identifying the framework's components is needed.
- From our experience using the hot-spot-driven approach, it became apparent that the guideline in respect of the

granularity of hot-spots of *about one method* (see [2]) is not precisely enough. It is not clear, how much functionality actually corresponds to each hot-spot. Therefore, we define the granularity of hot-spots and hot-spot-cards, respectively, to be *exactly one (hook) method* in the programming language. We support this decision by introducing so-called *group-hot-spot-cards*. A group-hot-spot-card comprises an arbitrary number of hot-spot-cards of the granularity of one hook method. With the original Pree process, the number of hook methods that belong to one hot-spot is only implicitly defined by the respective hot-spot-card. However, the concept of group-hot-spot-cards allows for explicitly determining the number of hook methods by the number of their hot-spot-cards. These group-hot-spot-cards are used to identify the framework's components and to specify the flexibility requirements of these components.

In the following, we present ProMoCF (short for "Process Model for Component Frameworks"), a modification of Pree's approach towards a lightweight process model for component frameworks. Figure 1 depicts an overview of ProMoCF. The single activities of this process model are described below:

Definition of a specific object model: Typically, component-based software development processes begin with defining the components of the application, e. g., [26, 27, 28]. In contrast, our process model for component frameworks begins with creating an initial object model (like the original Pree process). This initial object model is created on the analysis of example applications in the considered domain. We start with an initial object model, because the functionality of the component framework is evolved by stepwise abstraction from it. By this, also appropriate framework components emerge. If the considered example applications already base on component technology, also an initial component model can be created. In that case, the specific object model is accumulated by the object models of the components in the component model.

Identification of hot-spots and writing hot-spot-cards: The iterations of the process model's main cycle begin with the identification of hot-spots and writing of hot-spot-cards. Hot-spot-cards constitute a simple but effective means for documenting and communicating the flexibility requirements to the framework between the domain experts and the framework developers. By this, hot-spot-cards bring the specific knowledge of the domain experts into the framework development process. For examples of hot-spot-cards see [16, 17].

Grouping of hot-spot-cards and identification of components: In contrast to the original Pree process, the hot-spot-cards are arranged into logical groups and so called *group-hot-spot-cards* are written. Arranging the hot-spot-cards into logical groups requires a lot of cognitive work by the framework developers: The flexibility requirements described on each hot-spot-card are compared. Hot-spot-cards that describe flexibility requirements for a common task are arranged together and form a logical group. For each logical group, a corresponding group-hot-spot-card is written. For example, within a framework for managing files and folders, one would probably group the hot-spot-card to *Add an arbitrary document to a folder* with its counterpart, the hot-spot-card *Remove an arbitrary document from a folder*.[1] The corresponding group-hot-spot-card could be called *Manage arbitrary items in a folder*. It seems to be reasonable to realize each of the logical groups written down on the group-hot-spot-cards as a self-contained framework component. Con-

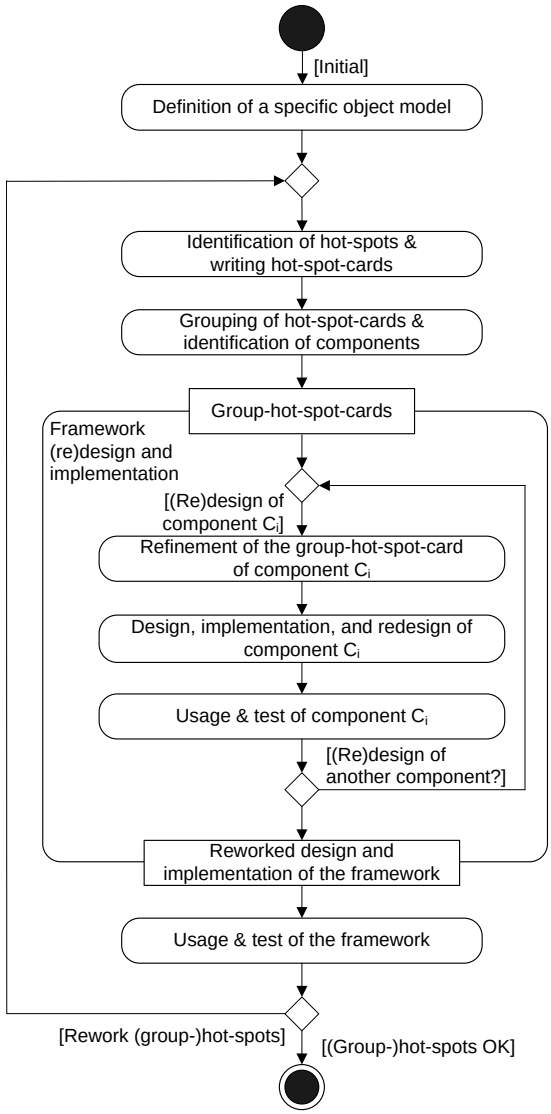

Fig. 1. The modified hot-spot-driven lightweight process model for component frameworks

[1] Note: In principle, hot-spot-cards can be divided into data-hot-spot-cards and function-hot-spot-cards. To be more precisely, the presented hot-spot-cards are data-hot-spot-cards (see [16]).

sequently, the flexibility requirements to the framework components are defined by their respective group-hot-spot-cards. By this, ProMoCF provides for developing flexible instances of the framework components. As we will see in Section 6, each group-hot-spot-card identifies one component within our component framework MM4U.

Grouping of hot-spot-cards to logical groups provides a means for abstracting from some flexibility requirements to the framework, if they are not necessary to discuss in detail. For example, not every single hot-spot-card needs to be discussed with the domain experts. By building logical groups of hot-spots and writing them down on group-hot-spot-cards automatically a coarser and with it a more concise view to the framework's flexibility requirements is created. The hot-spot-cards can be attached to their corresponding group-hot-spot-card, e. g., by using a paper clip. Also new group-hot-spot-cards can be added in this activity, for which no corresponding hot-spot-card could be determined in the previous activity so far, because it is not clear which hot-spots this group-hot-spot-card might have. Finally, group-hot-spot-cards can be build of other group-hot-spot-cards to provide an even more abstract view. As a consequence, nesting group-hot-spot-cards means to build a hierarchy of framework components.

The structure of group-hot-spot-cards is similar to the structure of hot-spot-cards. They have a name, a one-sentence description of what the group-hot-spot-card is about, a brief description of the group-hot-spot-card's behavior for at least two concrete situations, and two check boxes for determining whether adaptation during runtime is necessary and whether a configuration tool for end users is needed. Optionally, a list of the attached hot-spot-cards is added.

Framework (re)design and implementation: In this activity, the design and implementation of the component framework is conducted. The major difference to the original hot-spot-driven process is that it has sub-activities for developing the framework's components. For each component that shall be (re)designed and improved in the current main cycle iteration, the following sub-activities are conducted: First, further hot-spots of the component's group-hot-spot-card are identified. This results in a refined group-hot-spot-card. In the next sub-activity, the component is designed and implemented. For it, the component's interfaces are specified and at least one concrete instance of the component is implemented. This concrete implementation is tested in the next sub-activity by applying it in a test environment. The design and implementation of the component can be revised and improved in the following main cycle iterations. Implementing the components also includes writing the documentation that is needed for application developers to use the component. Since not all components are improved in each iteration, they can have different maturity. For example, while some components are in the implementation phase already, others can still be in the requirements or design phase. In addition, the development of the framework components can also be concurrently conducted (not shown in Figure 1). The result is a (revised) design and an (improved) implementation of the framework components.

Usage and test of the framework: In this activity, the (revised) framework is tested and evaluated in regard of the quality and reusability of its design. The only way to identify errors and weak points is to develop concrete applications that use the component framework [4]. Developing a concrete application means composing framework components (that meet the requirements of the application's domain) according to the rules defined by the framework. The framework components can be newly developed, but also created by, e. g., modifying existing ones or be reusing them in different configurations. Errors are imediately corrected and weak points in the framework design are improved in the next main cycle iteration.

In Section 4, we argued for the benefits of a lightweight process model for developing software frameworks. With ProMoCF, only (group-)hot-spot-cards and some source code documentation are written during the framework development process. In regard of documentation, only as much is written as needed to develop the framework and the concrete applications. Only for finishing a framework milestone, i. e., a release of the framework to an external organization, a complete documentation is written. ProMoCF has been applied to the development of the MM4U component framework for personalized multimedia applications. The development of this framework is described in the following section.

6. MM4U – A COMPONENT FRAMEWORK FOR PERSONALIZED MULTIMEDIA APPLICATIONS

The aim of the MM4U (short for "Multimedia for you") component framework [29, 30] is to simplify and to improve the development of personalized multimedia applications. It supports application developers during the design and implementation phase. The MM4U framework relieves application developers from the time-consuming multimedia composition task in order to let them concentrate on the development of the application-specific functionality. For it, application developers require extensive support for creating personalized multimedia content. Figure 2 illustrates the general creation chain for personalized multimedia content. In a first step, media elements such as images, text, audio, and video are selected according to the user profile information. To serve different target multimedia presentation formats, the media elements are assembled and composed in time and space using an internal multimedia representation model [29]. The representation model is designed to be easily transformed to different (standardized) multimedia presentation formats, e. g., SMIL [31], SVG [32], and Macromedia Flash [33]. This transformation is conducted in the subsequent task [30]. Finally, the presentation is delivered

to the client for rendering. In the following, we present the development of the MM4U framework along the single activities of the lightweight process model ProMoCF presented in Section 5.

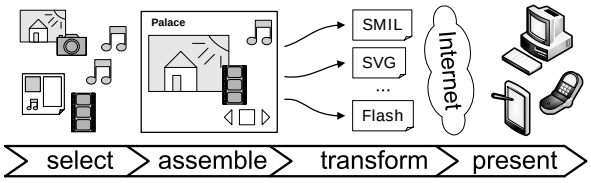

Fig. 2. The general multimedia personalization process

Analysis of related work and defining a specific object model: First, some general requirements to the framework were defined by conducting an extensive study of related work in the areas of user modeling, meta data modeling for media elements, multimedia composition, and multimedia presentation. Also other existing systems and approaches for generating personalized multimedia content were analyzed (see [29]). In addition, we analyzed prototypes of personalized multimedia applications that have been developed in different domains requiring multimedia personalization. These prototypes are a personalized city tour through Vienna [34], a GPS-based mobile paper chase game [35], and a personalized music news letter. On the basis of the extensive study of related work, the existing systems and approaches, and on the analysis of our prototypes, we defined the initial object model of the MM4U framework.

Identification of (group-)hot-spots and components: An initial set of hot-spots was identified and corresponding hot-spot-cards were written. For identifying the hot-spots, our experience in developing the prototypes for personalized multimedia applications as well as related work in the area of generating personalized multimedia content were helpful. The initial hot-spot-cards were arranged according to their description into logical groups and respective group-hot-spot-cards were written. Each of these group-hot-spot-cards addresses a particular task of the general multimedia personalization process depicted in Figure 2. During the iterative framework development process further hot-spot-cards were identified. These were attached to the existing group-hot-spot-cards. No further group-hot-spot-card was created.

In the end, five group-hot-spot-cards with their respective hot-spot-cards were identified for the MM4U framework. The first two group-hot-spot-cards specify the flexibility requirements needed for integrating existing systems and solutions for storage, retrieval, and access to user profile information as well as media elements with their associated meta data. The third group-hot-spot-card specifies the flexibility requirements for composing arbitrary personalized multimedia content in the internal multimedia representation model. The flexibility requirements for transforming the multimedia representation model to the syntax and features of different (standardized) multimedia presentation formats are written down on the fourth group-hot-spot-card. Finally, the fifth group-hot-spot-card specifies the flexibility requirements for integrating existing players for multimedia into the component framework.

The group-hot-spot-cards divide the flexibility requirements to the MM4U framework in five logical groups, each addressing a particular task in the multimedia personalization process. Consequently, we defined for each group-hot-spot-card a distinct component within the MM4U framework. We specified the interfaces of the framework components and implemented a concrete instance of these components. Each concrete implementation of the framework components is realized as an traditional object-oriented framework.

Only by identifying the group-hot-spot-cards, we were sure that all necessary framework components have been identified and that these are the right ones, i. e., that the flexibility requirements to the framework are reasonably divided into (the identified) components. As an example, the design of the multimedia composition component and presentation format generators component are presented in the following.

Design of the multimedia composition component: The multimedia composition component provides support for creating arbitrary personalized multimedia content in a tree-like fashion using the multimedia representation model mentioned above. It offers a set of application-independent basic composition elements. This set comprises media elements like `Image`, `Text`, `Audio`, and `Video`, temporal composition operators to determine the temporal course of the presentation, e. g., `Parallel` and `Sequential`, as well as projectors to determine the layout of the presentation, for example a `SpatialProjector` and an `AcousticProjector` for defining the spatial and acoustic layout of the presentation. Each of these composition elements is realized as a distinct class in the composition component. An application developer uses them as black-box and creates as much instances of these classes as needed to compose the personalized multimedia content. It is the responsibility of the constructors of these classes to actually create all necessary objects for composing the multimedia content.

For refining the composition component's group-hot-spot-card, the set of basic composition elements are not of further interest, since they constitute fixed elements for multimedia composition. However, the multimedia composition component should be extensible by more complex and application-specific composition functionality. This is realized by the composition interface `IComplexOperator`. We initially identified the hook method `doCompose(...)` as hot-spot of this interface (and as the only hot-spot of this component). However, the signature of this hook method is fixed, once it is defined in the framework. Concrete applications would not be able to pass application-specific parameters to their complex operators to realize the required

multimedia composition functionality. Thus, with refining the group-hot-spot-card, we changed the initial hook method to be the constructor of the concrete classes implementing `IComplexOperator`. The constructor is more flexible, since its signature can be defined individually for each concrete complex operator. In the following, we introduce the concept of complex operators and their enhancement, the sophisticated operators.

A complex operator encapsulates an arbitrary number of basic composition elements, i. e., media elements, temporal operators, and projectors. So, application developers can create more complex bricks for realizing their multimedia composition functionality. A complex operator always describes a static document structure, which means that the structure of the multimedia presentation is hard-wired within the operator. Consequently, sophisticated operators enhance the concept of complex operators by determining the structure and layout of the multimedia presentation during runtime. This is achieved by adding additional composition logic to the sophisticated composition operator in form of, e. g., program code, style sheets, or layout rules (for details see [29]).

Design of the presentation format generators component: For actually transforming the personalized multimedia content in the internal representation model to the syntax and features of the concrete presentation formats, the presentation format generators component provides an application-independent transformation algorithm. Starting with the root node of the personalized multimedia content tree, this algorithm traverses the tree and transforms the content representation to the concrete presentation formats (for details see [30]). The initially identified hot-spot-cards of this component's group-hot-spot-card where already sufficient and have one common task: Transforming multimedia content in the internal representation model to a concrete multimedia presentation format. Each hot-spot is responsible for transforming one of the basic composition elements. For example, the hook method `medium(…)` transforms the media elements, the methods `parallel(…)` and `sequential(…)` the temporal operators, and the methods `spatialProjector(…)` and `acousticProjector(…)` conduct the transformation of the projectors. Additional hook methods for transforming the complex and sophisticated composition operators are not necessary, since the abstract transformation algorithm automatically detects them and decomposes them into their basic bricks, i. e., the media elements, temporal operators, and projectors.

The multimedia composition component and presentation format generators component as well as the other framework components are not only integrated in our MM4U framework, but can also – according to the nature of components – be independently deployed. For example, the components for accessing user profile information and media elements can be used off-the-shell by arbitrary applications. We also set up a web service that only uses two components of the MM4U framework, the multimedia composition component and presentation format generators component. This web service can be used by applications that need to deliver their multimedia content in different target formats.

(Re)design, implementation, and usage of the MM4U framework: Many iterations have been conducted for (re)designing and implementing our MM4U framework. During these iterations, the single components had different maturity. For example, while implementing the multimedia composition component, the presentation format generators component was still in its initial design.

Using the MM4U framework for developing a concrete personalized multimedia application primarily means composing and reusing the already available implementations of the framework components. For example, the existing implementations of the presentation format generators component as well as the components for accessing the user profile information and media elements typically can be reused without modifying the components' source code, but by using them in different configurations only. However, the multimedia composition component is heavily dependent on the actual application's domain. Consequently, the concrete instance of this framework component changes with the considered domain. Developers of a personalized multimedia application typically implement their own multimedia composition component. They reuse the basic multimedia composition functionality the MM4U framework offers as well as any concrete composition and personalization functionality provided by other concrete applications based on our framework.

The MM4U component framework itself is designed as a component. Consequently, we created a sixth group-hot-spot-card for this component capturing the flexibility requirements of the MM4U framework on an higher abstraction level. This group-hot-spot-card merges the components of the MM4U framework into a single interface `IPersonalizedMultimediaApplication`. For using this component, application developers need to implement this interface and can execute their application by using the component's `MM4UApplicationRunner` class. This class captures the sequence of calling the methods that are defined in the `IPersonalizedMultimediaApplication` interface. The application runner class defines the order of using the MM4U framework components and how they interact.

The development of several demonstrator applications using the MM4U component framework, e. g., a personalized mobile city guide [36], provided further feedback for the framework development. These applications pinpointed the weak points of the framework's architecture, i. e., those parts of the design where too much or too less flexibility was implemented.

7. CONCLUSION

In this position paper we argued for a lightweight process model for developing component frameworks. We presented ProMoCF, a modification of the hot-spot-driven approach for object-oriented frameworks by Pree towards a process model for component frameworks. ProMoCF provides methodical support for identifying framework components and specifying the flexibility requirements to these components. This is achieved by introducing the concept of group-hot-spot-cards. A first evaluation of ProMoCF for the development of our component framework MM4U for personalized multimedia applications has been successful. In future, further evaluation studies are needed to foster our approach.

8. REFERENCES

[1] James Carey and Brent Carlson, *Framework Process Patterns: Lessons Learned Developing Application Frameworks*, Addison-Wesley, 2002.

[2] Wolfgang Pree, *Design Patterns for Object-Oriented Software Development*, ACM Press Books. Addison-Wesley, 1995.

[3] Clemens Szyperski, Dominik Gruntz, and Stephan Murer, *Component Software : Beyond Object-Oriented Programming*, Addison-Wesley, 2nd edition, 2002.

[4] Jan Bosch, Peter Molin, Michael Mattsson, PerOlof Bengtsson, and Mohamed E. Fayad, "Framework problems and experiences," In Fayad et al. [37], pp. 55–82.

[5] Ralph E. Johnson and Brian Foote, "Designing reusable classes," *Journal of Object-Oriented Programming*, vol. 1, no. 2, pp. 22–35, June/July 1988.

[6] Rebecca J. Wirfs-Brock and Ralph E. Johnson, "Surveying current research in object-oriented design," *Communications of the ACM*, vol. 33, no. 9, pp. 104–124, 1990.

[7] Jan Bosch, *Design and Use of Software Architectures : Adopting and evolving a product-line approach*, Addison-Wesley, 2000.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design patterns : elements of reusable object-oriented software*, Programmer's Choice. Addison-Wesley, July 2004.

[9] Don Roberts and Ralph Johnson, "Evolving frameworks – a pattern language for developing object-oriented frameworks," in *Pattern Languages of Program Design 3*. Addison-Wesley, Illinois, USA, 1997.

[10] Wolfgang Pree, "Frameworks – past, present, future," *Object magazine : improving software quality through object development & reuse*, vol. 6, no. 3, May 1996.

[11] Wolfgang Pree, "Component-based software development – a new paradigm in software engineering?," *Software – Concepts and Tools*, vol. 18, no. 4, pp. 169–174, 1997.

[12] Wolfgang Weck, "Independently extensible component frameworks," in *Special Issues in Object-Oriented Programming*, M. Mühlhäuser, Ed., Heidelberg, 1997, pp. 177–183, dpunkt.verlag.

[13] Mohamed E. Fayad, Douglas C. Schmidt, and Ralph E. Johnson, Eds., *Implementing Application Frameworks: Object-Oriented Frameworks at Work*, John Wiley & Sons, 1999.

[14] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford, *Documenting Software Architectures : Views and Beyond*, Addison-Wesley, 2003.

[15] Marcus Fontoura, Wolfgang Pree, and Bernhard Rumpe, *The UML Profile for Framework Architectures*, Object Technology Series. Addison-Wesley, 2002.

[16] Wolfgang Pree, *Framework Patterns*, SIGS Books and Multimedia, 1996.

[17] Wolfgang Pree, "Hot-spot-driven framework development," In Fayad et al. [37], pp. 379–393.

[18] K. Beck and W. Cunningham, "A laboratory for teaching object oriented thinking," *ACM SIGPLAN Notices*, vol. 24, no. 10, pp. 1–6, 1989, OOPSLA'89 Conference Proceedings.

[19] Hans Albrecht Schmid, *Framework Design by Systematic Generalization*, chapter 15, pp. 353–378, In Fayad et al. [37], 1999.

[20] Dirk Riehle, *Framework Design: A Role Modeling Approach*, Ph.D. thesis, Swiss Federal Institute of Technology Zurich, 2000.

[21] Niklas Landin and Axel Niklasson, "Development of object-oriented frameworks," Diploma thesis, Lund Institute of Technology, Lund University, Lund, Sweden, 1995.

[22] Kent Beck, *Extreme programming explained: embrace design*, Addison-Wesley, Dezember 2000.

[23] Cockburn, Alistair, "Crystal main foyer," http://alistair.cockburn.us/crystal/crystal.html, 2005.

[24] Philippe Kruchten, *The Rational Unified Process : an introduction*, Addison-Wesley Object Technology Series. Addison Wesley, 2nd printing edition, 2000.

[25] Richard N. Taylor, Will Tracz, and Lou Coglianese, "Software development using domain-specific software architectures," *SIGSOFT Softw. Eng. Notes*, vol. 20, no. 5, pp. 27–38, 1995.

[26] Ian Sommerville, *Software Engineering*, Pearson and Addison Wesley, 7th edition, 2004.

[27] Jim Q. Ning, "A component-based software development model," in *20th Computer Software and Applications Conference*, Seoul, Koria, 1996, pp. 389–395, IEEE Computer Society Press.

[28] Ali H. Dogru and Murat M. Tanik, "A process model for component-oriented software engineering," *IEEE Software*, vol. 20, no. 2, pp. 34–41, 2003.

[29] Ansgar Scherp and Susanne Boll, "MM4U – A framework for creating personalized multimedia content," in *Managing Multimedia Semantics*, Uma Srinivasan and Surya Nepal, Eds., chapter 11. IRM Press, 2005.

[30] Ansgar Scherp and Susanne Boll, "Paving the last mile for multi-channel multimedia presentation generation," in *Procedings of the 11th Multimedia Modeling Conference*, Melbourne, Australia, Jan. 2005, pp. 190–197, ACM Press.

[31] W3C, "Synchronized Multimedia Integration Language (SMIL 2.0) Specification," 2001, W3C Recommendation 08/07/2001 - http://www.w3.org/TR/2001/REC-smil20-20010807/.

[32] W3C, "Scalable Vector Graphics (SVG) 1.2 Specification," 2004, W3C Working Draft 05/10/2004 - http://www.w3.org/TR/2004/WD-SVG12-20040510/.

[33] Macromedia, Inc., USA, "Macromedia – Flash MX 2004," 2004, http://www.macromedia.com/software/flash/.

[34] Susanne Boll, "Vienna 4 U – What Web Services can do for personalized multimedia applications," in *Seventh Multi-Conference on Systemics (SCI 2003), Cybernetics and Informatics*, July 2003.

[35] Susanne Boll, Jens Krösche, and Christian Wegener, "Paper chase revisited – a real world game meets hypermedia," in *Proc. der Intl. Conference on Hypertext (HT03)*, Aug. 2003.

[36] Ansgar Scherp and Susanne Boll, "Generic support for personalized mobile multimedia tourist applications," in *ACM Multimedia, Technical Demonstration*, Oct. 2004.

[37] Mohamed E. Fayad, Douglas C. Schmidt, and Ralph E. Johnson, Eds., *Building Application Frameworks : Object-Oriented Foundations of Framework Design*, John Wiley & Sons, 1999.