2009 Seventh International Conference on Creating, Connecting and Collaborating through Computing

Sophie—Tools and Materials in Multimedia Book Creation

Norman Holz*, Robert Hirschfeld*, Jens Lincke*, Michael Haupt* and Michael Rüger†

*Hasso-Plattner-Institut, University of Potsdam
Potsdam, Germany
{norman.holz, hirschfeld, jens.lincke, michael.haupt}@hpi.uni-potsdam.de

†Impara GmbH
Magdeburg, Germany
m.rueger@acm.org

Abstract

Sophie is an authoring tool for interactive multimedia books. The Sophie user can combine different kinds of media and synchronize them with events and time. Such highly interactive systems are difficult to design and to implement. The large amount of functionality provided has to be balanced with a significant ease of use. The standard software engineering approaches for the construction of large software systems are insufficient concerning the special requirements of interactive systems. The Tools and Materials pattern language is an alternative approach to design interactive systems. This paper exemplifies the use and benefits of Tools and Materials by the Sophie application.

1. Introduction

What distinguishes your favorite application from the alternatives? Interactive software systems are designed and built to help users solving their problems. Often the users can choose between several applications to accomplish specific tasks. The users will prefer the application that provides the most efficient way to solve their problem. Thus the preferred application gives competence to accomplish certain tasks in a comfortable and efficient manner.

Sophie is an authoring application for multimedia books [1]–[3]. It unites the features of a word processor and an application like Adobe's Director. Sophie users can combine text with several kinds of media, such as images, movies, and audio streams. Media can be synchronized with time and events to create an interactive multimedia book. Sophie is designed to be intuitively and comfortably used without any special knowledge or even programming skills.

Sophie is developed by impara Magdeburg [4]. The core team of five developers started their work in summer 2004 and Sophie 1.0 was released recently. Sophie is open source, implemented in Squeak [5], and available for Mac OS, Linux and Windows.

The field of software engineering provides several approaches to designing and realizing large software systems as Sophie. A very common one is the 3-Tier Architecture. This approach divides the application into three main layers. At the bottom level, the application is based on a data layer responsible for persistency and data management. The application layer comprises the objects altering these data. On top, the user interface layer describes everything related to the user interface. This approach is general enough to fit on almost every larger software system. In order to think and speak about an interactive software system appropriately, a more detailed point of view is required.

A finer-grained approach is the Tools and Materials pattern language, developed by Dirk Riehle and Heinz Züllighoven [6]. This paper introduces the Tools and Materials pattern language in general and exemplifies the practical use of the abstract principles by means of Sophie. As a conclusion the benefits of Tools and Materials for the design of interactive software systems are elaborated.

2. Tools and Materials Pattern Language

In this section, the Tools and Materials pattern language is introduced generally. The most important design metaphors and design patterns are discussed in order to establish a formal basis for the examination of the Sophie application.

The pattern language is based on the Tools and Materials metaphor. This metaphor is an approach with a specific underlying view of human work [6]–[8]. The main idea is characterized by craftsmen using tools to work on materials in an environment. They have the necessary skills and organize their work and environment accordingly.

This metaphor is transferred to the field of software engineering and the construction of large software systems by Dirk Riehle and Heinz Züllighoven by means of their pattern language. Its basis is provided by fundamental design metaphors, which are described in the following.

978-0-7695-3620-0/09 $25.00 © 2009 IEEE
DOI 10.1109/C5.2009.16

2.1. Design Metaphors

Design metaphors are defined by Dirk Riehle and Heinz Züllighoven as patterns governing the perception of the application domain and as a guide for designing the future system.

Tools and Materials' central notions are the two design metaphors "Tool" and "Material". They describe a conceptual distinction between materials as things to be worked upon and tools as the means of work. Tools are considered to be active entities, providing information about their current state and giving feedback to the users. Materials are passive entities, which are modified and transformed during work. Materials can be accessed via tools only.

Often one tool can be used to work on several materials and a specific material can be treated in different ways by different tools. The Tools and Materials pattern language uses the concept of "Aspects" to make the relationship between tools and materials explicit. Aspects define the necessary operations to work properly with materials concerning specific tasks. Thereby they provide an abstraction from concrete tools and concrete materials. Based on the design metaphors Tool, Aspect, and Material, the pattern language provides certain design patterns, which we will discuss next.

2.2. Design Patterns

Design metaphors are useful to perceive and design the future system, whereas design patterns provide a kind of "micro architecture" in the technical construction process [6]. The Tools and Materials pattern language distinguishes between patterns for constructing tools and patterns covering the tool integration into the environment. In the following, the different design patterns are introduced and visualized by the notation shown in Fig. 1.

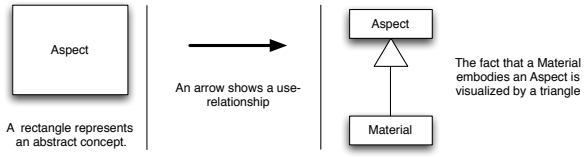

Figure 1. General notation.

2.2.1. Tool Construction. The future system can be seen as a tool itself, consisting of subtools comprising further subtools. This hierarchic structure corresponds to the Tools and Materials design pattern of Tool Composition and helps to split up the whole system in several subparts with related responsibilities and implementation issues. The idea of Tool Composition is very close to the universal principle of divide and conquer in software engineering.

A tool works on a set of materials. As already mentioned above, the relationship between tools and materials is defined by aspects exclusively, which relates to the design pattern of Tool and Material Coupling. Most materials embody several aspects and therefore several tools can work on them. Due to this indirect coupling mechanism, different tools can work on the same material in the context of the same aspect. A tool implementing a certain aspect can be used to work on several materials relating to this aspect. Generally, this approach allows a flexible composition of tools to build up the system.

At the bottom level, a compound tool comprises simple tools that do not consist of further subtools but implement a specific functionality. For the appropriate construction of simple tools the pattern language defines the pattern Separation of Powers. This pattern divides the simple tool into a functional and an interaction part. The functional part alters the material according to the material's aspect, whereas the interaction part manages the tool's user interaction. With respect to the Tool design metaphor the interaction part presents the material and informs the users about the tool's current state. Additionally the users will get feedback to their recently performed actions on the material. This design pattern can be mapped to the Model-View-Controller pattern [9]. The interaction part acts as the View as well as the Controller, whereas the functional part together with material and aspects is considered as the Model.

Both parts need to inform each other about relevant changes. This communication should not result in directly coupling the two components. Preferably, an event-based communication mechanism, e. g., using the Observer pattern [10], should be used instead. This loose coupling provides the possibility to have several interaction parts working with one functional part or to change the implementation of the functional part independently. Figure 2 gives an overview of the collaboration of the design patterns for tool construction.

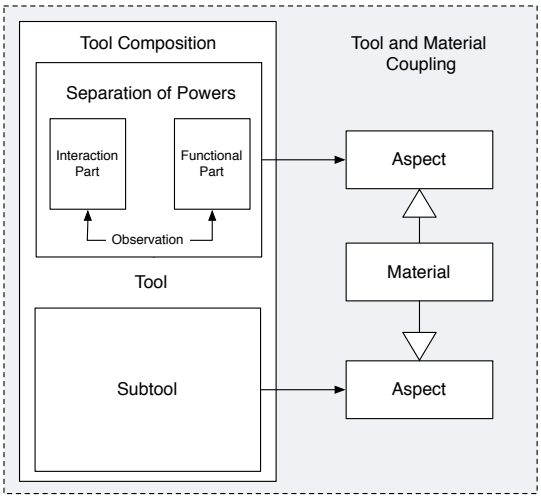

Figure 2. Patterns for Tool Construction [6].

In addition to the design patterns for tool construction, the pattern language defines further design patterns to integrate

the constructed tools to built up the whole system. These design patterns for tool integration are described in the following.

2.2.2. Tool Integration. The place where tools, materials and their collaboration are organized is described by the Environment design pattern. The environment creates, initializes and presents the tools and materials to the users, waiting for them to start working.

Besides the presentation of accessible materials in the environment, it has to be clarified by whom the materials are supplied and where the materials are kept after manipulation. This is realized by the design pattern of the Material Administration. The material administration supplies the material to the environment and manages the persistency. For this purpose, it uses several Material Providers to offer materials from different sources.

In complex systems, certain dependencies may exist among related materials. In order to maintain the loose coupling of tools and the system's flexibility, material dependencies are managed at the material level, instead of at the level of tools. The Material Container design pattern defines a container that collects the dependent materials and maintains the material constraints.

After this introduction to the concepts of Tools and Materials, their use is illustrated by the example of Sophie in the next section.

3. Tools and Materials in an Interactive System

Sophie is an interactive system with a large number of features. In the context of the Tools and Materials pattern language, Sophie is considered a very powerful tool to work on the material of multimedia books. Sophie uses the Tools and Materials approach both at the level of implementation and the level of user interface design.

3.1. Tools and Materials in User Interface Design

At the user interface level, the natural metaphor of Tools and Materials helps the users to understand the system and to accomplish their tasks intuitively.

Sophie uses a workspace to provide a working environment for its users [1]. Within this workspace, Sophie presents the currently opened Sophie book as the material and gives access to different functionalities embodied by the corresponding tools.

Besides the pure functionality necessary to create media-rich books, Sophie is designed to be used in an easy and intuitive manner. Although an application offers a large number of powerful functionalities, users will not use them if they cannot find the corresponding tools or have to access them in an inconvenient way. In order to make user interaction with Sophie comfortable, the Sophie user interface design follows intuitional ways to present the available tools and materials.

General tools—for instance the spell checker, and the materials the Sophie book embeds—are collected in flaps [1]. Flaps, as shown in figure 3, can be seen as drawers within the user's tool box containing related objects. These drawers can be opened and closed on demand.

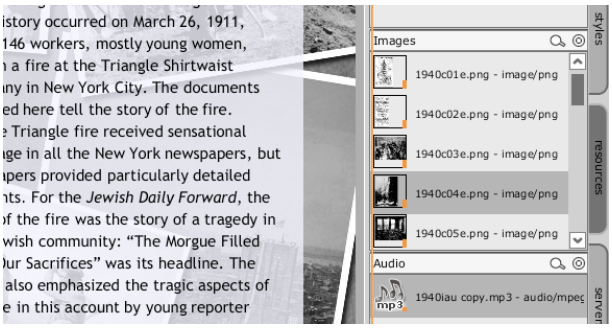

Figure 3. Flaps containing objects.

Sophie users start creating a Sophie book by dragging a book template from the library flap into the workspace. Then they change to the resource flap and add a picture or movie to the Sophie book. Finally, the users get an overview of the book's pages in the pages flap and can check the spelling via the spell checker accessible via the tools flap. Due to the fact that the tools assembled in an opened flap and the titles of closed flaps are always visible, users get a quick view on the tools and materials available. Therefore they do not have to inconveniently search in nested structures of menus and modal dialogues.

Sophie gives access to tools corresponding to a specific material item, for instance a piece of text or an embedded image, by the means of halos and head-up displays (HUD) [1]. Halos are small icons that show up close to the selected object. As an example, a piece of text can be altered in terms of its font and layout. Both aspects are displayed to the users in the form of a small halo button. On mouse click, these halo buttons give access to a HUD that offers the tools suitable in this context. Often only a subset of the offered tools is used frequently, therefore the HUDs show the very common functionality at first and can be expanded to access the more special and thus rarely used tools on demand. Figure 4 on the next page shows the halo buttons used to alter the text's font settings.

HUDs are a direct realization of the Tool metaphor. The font HUD, shown in figure 5 on the following page, provides state information, for instance the current font size. It gives feedback to recent user actions by updating the displayed values accordingly. Due to that spatial affiliation of tool and material, the material itself and the performed changes are comfortably presented to the users.

Figure 4. Halo buttons.

Figure 5. Head-up displays.

3.2. Tools and Materials in Implementation

When examining Sophie's architectural design in the context of Tools and Materials, the application is structured in several subtools. As an example, the tool collections responsible for media and resource management, or the tool chain which builds the actual pages of a Sophie book, can be identified easily. In the following, these parts of the Sophie application are set into the context of the pattern language to exemplify the use of the introduced concepts for implementing an interactive system. For this purpose, implementation examples of Sophie are chosen and visualized by an extended notation. The elements added to the general notation are shown in figure 6.

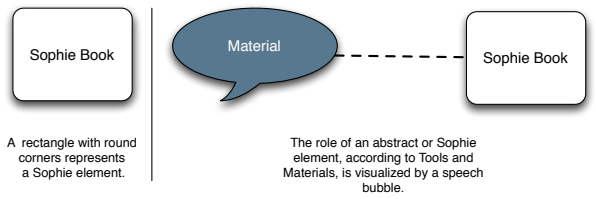

Figure 6. Notation elements for the Sophie examples.

Sophie deals with different kinds of media. Besides the media stored on the local file system, Sophie is able to embed remote audio, video and image files in a Sophie book. The Sophie Resource Manager [1] is responsible for media retrieval and storage, relating to the design pattern Material Administration [6]. It uses several material providers, each responsible for a different kind of source, to provide transparent access to the local file system, a remote server, or even further sources such as an SQL database.

The embedded media of a Sophie book can be transformed in many different ways. For instance, images can be cropped, rotated, or scaled. Media transformation in Sophie is generally implemented in the Transformation Manager [1]. From the Tools and Materials' point of view, the Transformation Manager can be seen as the tool used to transform the media material. Due to the fact that the Transformation Manager should be able to deal with a great variety of media, it consists of several subtools implementing specific transformation routines for specific media objects, such as image rotation or scaling video clips, as shown in figure 7 on the following page.

This implementation of the Tool Composition design pattern provides the possibility to treat functionality encapsulated in tools independently from the rest of the system. Sophie takes advantage of this mechanism to provide different environments for authors and readers. In the authoring environment the users are able to edit the Sophie book and its components, whereas the reading environment is used to read and interact with the Sophie book. Therefore, an embedded video clip will start playing on mouse click in the reading environment, instead of being selected for editing. With respect to these different environments, the text editing functionality is encapsulated in a single tool. Therefore, the only action required to change from authoring to reading environment, is to disable or remove the text editing tool from the system.

When examining complex processes in Sophie with several components participating, the Tools and Materials pattern language is again very intuitive and natural. Generally, Sophie is used to create multimedia books. The media material provided by the resource manager is combined with the text and style information material into a book page by the page compositor tool. Book pages themselves are the material a page renderer tool can work on. Since a Sophie book can be displayed on-screen as well as be printed on paper, two different tools, the screen renderer and the postscript renderer, are used to work on the same material concerning the rendering aspect. This mechanism, shown in figure 8 on page 6, relates to the loose coupling of tool and material via aspects.

4. Example: Text Editor with Auto Completion

To illustrate the benefits of the acquired understanding of the Sophie system, we want to extend the text editing tool already mentioned in section 3, and add an auto completion functionality. Auto completion should be able to remember a typed word and propose it again when we start to type its first characters again. The proposal should disappear when

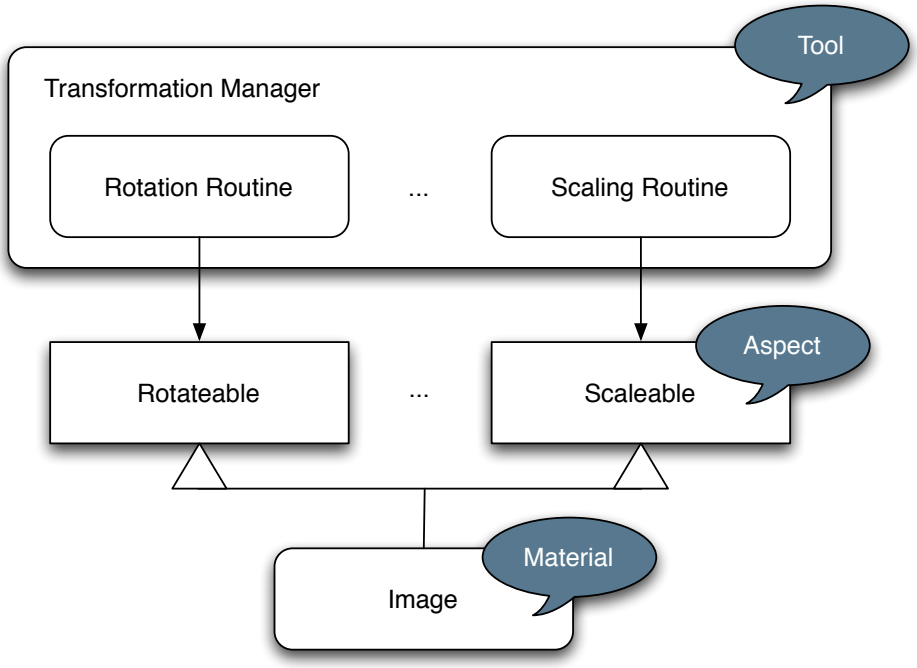

Figure 7. Tool Composition in the Transformation Manager.

it recognizes a character sequence that does not match any of the stored words.

As already described the text editing functionality is encapsulated in a single tool. Therefore our changes will only affect this specific tool and the integration of our auto completion functionality will be very straightforward.

As Sophie is implemented in Smalltalk, an object-oriented language, adding further functionality while keeping the original behavior can be done easily by means of inheritance. Our extended text editor inherits the basic functionality from the original one and is extended by a memory object that is able to store typed character sequences.

As shown in listing 1, the typed character is sent to the original text editor, and additionally logged and appended to the `currentWord`. If it is a separator, such as a space character, we know the current word is typed completely and remember it. If we exceed a word length longer than four we look for a word starting with this character sequence in our memory.

In listing 2, the logged key strokes are sent to the original text editor again, bypassing the logging mechanism, if a matching entry is found in our memory. If not, we store the character sequence ended by a separator in our memory. As shown in listing 3 on the following page, the implementation of `rememberCurrentWord` reasonably stores only words exceeding a specified length to prevent auto completion from constantly proposing very short completions.

Due to Sophie's intuitive design using the Tools and Materials approach, its open-source implementation in Smalltalk and our understanding of the text editor tool, we could easily extend the original tool with a convenient auto completion. We have simply inherited from the original text editor tool and added a few lines of code, most of them shown in the

```
insertIn: pagePlayer character: keyEvent

 super insertIn: pagePlayer character: keyEvent.

 (keyEvent keyCharacter isSeparator)
  ifFalse: [

    "log letter keystrokes"
    self currentWord add: keyEvent.

    "Search current word in memory"
    (currentWord size > 4)
     ifTrue:
       [self thinkAboutCurrentWord: pagePlayer].]

  ifTrue: [self rememberCurrentWord]
```

Listing 1. Remembering the typed word.

```
thinkAboutCurrentWord: pagePlayer

 "search for a matching word in memory"
 (self memory match: self currentWord)
  ifTrue: [
    self insertIn: pagePlayer memorizedEntry:
      (self memory matchingEntryFor: currentWord)]
```

Listing 2. Thinking about the current word.

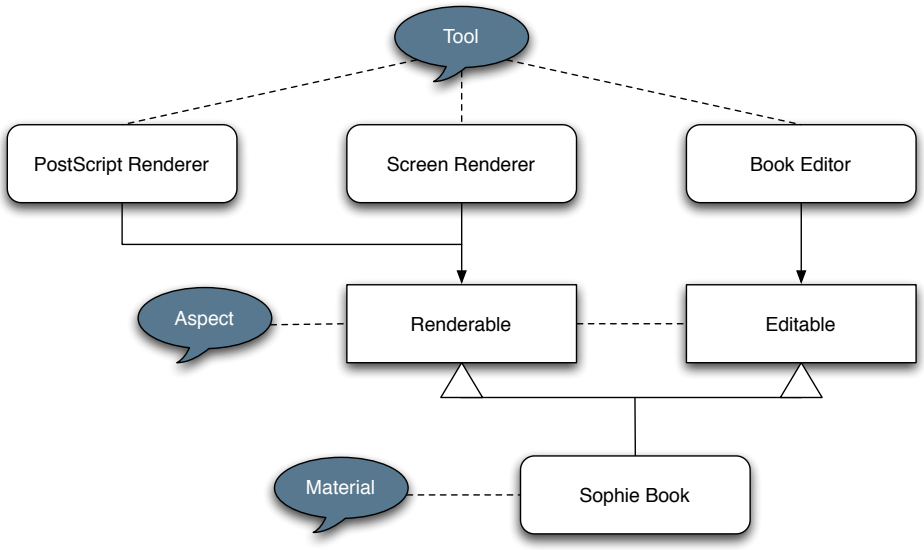

Figure 8. Tools for a Sophie book.

```
rememberCurrentWord

  "remember current word permanently"
  (self currentWord size > 10)
    ifTrue:[self memory add: currentWord].

  "reset the current word log"
  self forgetCurrentWord
```

Listing 3. Remembering the current word.

listings, without affecting further parts of Sophie.

5. Conclusion

The construction of large software systems is a difficult and complex process. The field of software engineering provides several approaches and concepts to handle that complexity and to implement a system that works correctly. Developing an interactive system meant to interact with users to support them in accomplishing certain tasks is even more complex and error prone. Even though a system may work correctly, it may still fail in the field of user experience and usability if it does not embrace suitable concepts to implement and to offer the possible very large number of expected features. The application people preferably use is either the only application available or it is the most comfortable tool that makes people feel competent to accomplish their tasks efficiently.

The Tools and Materials pattern language developed by Dirk Riehle and Heinz Züllighoven is a fine-grained approach to construct interactive software systems. It refines the coarse-grained 3-Tier Architecture by giving general design metaphors to think and speak about the system. For the system's implementation, several design patterns are provided. These refine commonly acknowledged software design patterns and best practices, such as the Model-View-Controller and Observer Patterns, and set them into the context of building an interactive system.

The Tools and Materials metaphor helps to design an interactive system at the level of implementation as well as at the user interface level. According to the implementation the developer will be able to understand the system as a whole and can easily determine where to search for the details. The actual users will again be able to understand the application as a whole with the perception of a richly filled tool box. With this, users can flexibly organize their workflows and working environments in order to accomplish their tasks in an efficient and comfortable manner.

Sophie is such a toolbox. As shown in this paper, Sophie is a feature-rich and easy to use authoring software for multimedia books, based on the Tools and Materials metaphor. Due to its strengths, Sophie will be able to compete with upcoming alternative applications in the competition for the users' preferences.

References

[1] "Sophie Project," http://www.sophieproject.org/. [Online]. Available: http://www.sophieproject.org/

[2] "The Institute for the Future of the Book," http://www.futureofthebook.org/. [Online]. Available: http://www.futureofthebook.org/

[3] M. Rüger, B. Stein, and D. Visel, "Sophie—The Future of Reading," in *Proceedings of the Sixth International Conference on Creating, Connecting and Collaborating*

through Computing (C5 '08), R. Kadobayashi, H. Kita, and R. McGeer, Eds. IEEE, 2008, to appear.

[4] "impara Magdeburg," http://www.impara.de/. [Online]. Available: http://www.impara.de/

[5] D. Ingalls, T. Kaehler, J. Maloney, S. Wallace, and A. Kay, "Back to the future: the story of squeak, a practical smalltalk written in itself," in *OOPSLA '97: Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, NY, USA: ACM Press, 1997, pp. 318–326.

[6] D. Riehle and H. Züllighoven, "A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor," in *Pattern Languages of Programm Design*, J. O. Coplien and D. C. Schmidt, Eds. Addison-Wesley, 1995, ch. 2, pp. 9–42.

[7] D. Riehle, *Entwurfsmuster für Softwarewerkzeuge*. Addison-Wesley, 1997.

[8] D. Riehle, B. Schäffer, and M. Schnyder, "Design of a Smalltalk Framework for the Tools and Materials Metaphor," *Informatik/Informatique*, vol. February, pp. 20–22, 1996.

[9] G. E. Krasner and S. T. Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80," *Journal of Object-Oriented Programming*, vol. August/September, pp. 26–49, 1988.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.