Design of a Smalltalk Framework for the Tools and Materials Metaphor

Dirk Riehle, Bruno Schäffer and Martin Schnyder

{riehle, schaeffer, schnyder}@ubilab.ubs.ch
Union Bank of Switzerland, UBILAB
CH-8021, Zürich, Switzerland

Abstract

The Tools and Materials Metaphor is a design methodology for interactive software systems. It supports developers with both metaphors on how to interpret and understand an application domain as well as concrete techniques on how to actually construct systems according to the metaphors. The construction process is best supported by an application framework that captures those parts of the metaphors that can be formalized as reusable abstract classes. We have implemented such a framework for the Tools and Materials Metaphor in Smalltalk.

1 Introduction

Building interactive software systems is a difficult task. Many approaches exist that try to deal with the arising issues. Some approaches address what kind of software is to be developed and how it looks like, which includes organizational, social, human-factor and user-interface issues. Other approaches address how the system can be built, which includes basic as well as advanced techniques, for example object-orientation as well as visual composition, frameworks and componentware. Only few approaches, however, try to integrate these what and how questions of software development. As a consequence, developers often have to bridge wide gaps between analysis, design and implementation techniques.

The Tools and Materials Metaphor [BCS92] is a methodology used to develop interactive software systems to support work in offices and workshop like settings. It has been successfully used in industrial projects, the largest one of which consists of a series of 6 banking projects totaling more than 1800 classes [BGZ95].

The metaphors of tool, aspect and material (and some others more) guide the developers during analysis, design and implementation activities and help them to perceive the system as an integrated whole on all relevant levels. On the software design level, the approach offers several concepts that have recently been elaborated as design patterns [RZ95]. Based on these concepts, software design can best be supported by a framework. Until recently, all frameworks to support software designed according to the Tools and Materials Metaphor have been developed in C++.

At UBILAB, the information technology laboratory of the Union Bank of Switzerland, we have developed a framework for the Tools and Materials Metaphor using IBM Smalltalk [IBM94]. In this paper, we present the result of this process, viz. discussions of:

- the Tools and Materials Metaphor,
- what extensions to the basic infrastructure offered by IBM Smalltalk (change/update protocol and metalevel support) were necessary,
- the design of interactive tools and their connection with materials,
- when and why we think multiple inheritance is needed and how we deal with this in a single inheritance system by means of additional tool support.

Section 2 presents the Tools and Materials Metaphor, section 3 the enhanced framework infrastructure support, section 4 the construction of interactive tools and section 5 the issues arising in coupling tools with materials via multiple inheritance. Section 6 draws some conclusions.

2 The Tools and Materials Metaphor

The Tools and Materials Metaphor can best be described in terms of a leitmotif, metaphors and design patterns used to implement the metaphors:

- A leitmotif explains why and what kind of systems we intend to build.
- The metaphors are high-level concepts used to interpret an application domain and to envision a concrete software system.
- Design patterns help to design and implement a system on a technical level.

The leitmotif of the Tools and Materials Metaphor is to develop software for the working place to support skilled human work, carried out by knowledgeable experts that can take over the responsibility for their work. This lends its application domain to skilled office work (e. g. in a bank) and workshop like settings.

Work in these domains can be interpreted using metaphors. Work is performed by using *tools* and *materials*. People use tools as a means of work to access and manipulate materials, the intermediate products and outcome of work. We interpret a typewriter as a tool that is used to fill out a form which is a material. Just like this we interpret a word processor (a tool) used to write a paper like this (a material). The distinction of tools and materials is one of the key ideas of the Tools and Materials Metaphor that permeates the analysis, design and implementation level.

The distinction of tools from materials is not to be mistaken as just another variant of view from model separation. Traditional approaches like the separation of view and model have only focussed on software design and are based on software engineering principles. It is hardly possible to talk with users about a word processor (window) being a view on a paper being a model. It is easy, however, to talk with them about a word processor being a tool they use to work on a paper being a material.

Next to the metaphors of tool and material, the metaphors of aspect and environment are used. An aspect denotes a specific way of handling a material from a tool's point of view. For example, a typewriter filling out a form makes use of its ability to be filled out. This might include some basic get and set operations as well as more complex check operations on the form values. The functionality needed for a specific task, here filling out, is called an aspect. It describes a tool's perspective on a material and the functionality of a material required by the tool. Generally many different kinds of materials can be viewed through the same perspective of a particular aspect.

A material can usually have many aspects. For example, a word processor document may have some simple aspects like being listable in a file dialog or being searchable for certain text patterns and a very elaborate aspect that lets a tool perform complex text formatting tasks on the document. A tool that makes only use of an aspect like searchable can work on any material that provides the functionality of being searchable, not just documents.

Finally, we organize our tools and materials in a working environment which offers some spatial and any number of logical dimensions to us. Today, the notion of environment is usually mapped to the notion of a computer desktop.

Further information on the Tools and Materials Metaphor can be found in [KGZ92, BGZ95].

3 Basic Infrastructure

In the design and implementation of the framework we had to enhance both the change/update protocol and the metalevel facilities available.

3.1 Extensions to Change/Update Protocol

In Smalltalk the change/update protocol allows objects to register themselves as dependents via the message addDependent:. Objects notify their dependents about state changes via the message changed: leading to an update: message.

IBM Smalltalk offers only a very basic update: implementation which passes a simple object interpreted as a constant around. Since any reasonable framework requires a more enhanced update facility than this, we enhanced the update: operation to pass a pair around consisting of an event symbol and the sender of that object.

By contrast the change/update protocol in VisualWorks\Smalltalk [Par94] is even more flexible. With the update: message it allows to pass an aspect (usually a symbol), a parameter containing additional information, and the sender of the changed: message.

3.2 Extensions to Metalevel Architecture

Systems based on the Tools and Materials Metaphor might become large and complex and have to be highly configurable. We therefore introduced a simple specification mechanism for classes [Rie95].

One use of these specifications is to retrieve sets of classes that offer the specified properties, for example all tool classes that can be instantiated, or all strategy classes that can sort a material according to different criteria, or all material classes that offer a certain aspect.

Another use is to specify a class by means of another object its instances have to work with. For example, given a material, we retrieve the default tool class for it. Or given the functional part of a tool component, we retrieve the default interaction part for it (see Tool Construction below).

Since the specification mechanism never directly refers to specific classes but only to abstract superclasses, subclasses can be changed without affecting the clients making use of them. We simply configure system variants by supplying certain tool classes or leaving them out.

4 Tool Construction

Tools designed according to the Tools and Materials Metaphor exhibit a regular structure and are therefore a fine target for framework support. Several issues characterize the structure and dynamics of a non trivial tool:

- Tools are built from *tool components*. Tool components may either be *simple* or *complex*. If they are simple, they do not rely on further tool components. If they are complex, they use further tool components as *subtool components*. The resulting structure of tool components constituting an overall tool corresponds to a single rooted tree much like the Composite pattern in [GHJV95].
- A complex tool component not only consists of some subtool components, but also provides the context of interpretation for them. It becomes the *context tool component* for them. A context tool component creates its subtool components, uses them according to its needs, interprets its results within its own context and deletes them if necessary. This is the pattern of Tool Composition in [RZ95].
- Each tool component usually holds one or more materials which it presents to the user and lets him work on. It might share this material with its context and/or subtools. It might as well supply its subtools only with parts of the overall material handled by the tool.
- A user may usually interact freely with any tool component of an overall tool. As a consequence, a context tool component has to be notified of state changes of subtool components and its materials. To do this, a context tool component observes its subtools and is notified of changes (this is the Observer pattern in [GHJV95]).
- Furthermore, each tool component may propagate a subtool component's state change to its own context tool component until a component is reached that has sufficient knowledge of context to appropriately interpret the state change (this is the Chain of Responsibility pattern in [GHJV95]).
- A tool is concerned with implementing both user interaction facilities and actual tool functionality. Every tool component is therefore subdivided into an interaction part implementing a tool component's particular interactive handling properties and the presentation of materials, and a functional part that implements the actual functionality of the tool to inspect, access and manipulate materials.
- As an addition to the separation of interaction from function we have introduced a user interface part. It implements the widget structure representing the interaction state of a tool component. This separation of user interface part from interaction part keeps classes generated and manipulated by a user interface builder apart from the behavior associated with this class.
- Since each tool component consists of one or more interaction parts and one functional part, a dual hierarchy of interaction and functional parts results. The power to decide about subtool components however resides always within the functional part.
- At the top of a tool component hierarchy, a single functional part and one or more interaction parts can be found. We wrap this top-level tool component by an additional tool object that represents the tool closure to the surrounding environment, that is the desktop.

In a running system, all tools are presented as icons on a desktop and managed by a central environment object which creates and deletes them according to a user's command. The environment object represents the overall systems closure and is the first object to be created during system startup.

5 Aspects and Materials

As mentioned in section 2, tools work on materials using aspects. An aspect is expressed as a class interface, by so called aspect classes. An aspect class represents the formalized functionality of a material required by a tool to perform the intended tasks.

A tool might use one or more aspect classes. For very complicated ways of handling a material the full material interface might be used as well, but this is usually an exception.

Material classes are application domain specific. Aspect classes in turn are more general since several ways of handling materials span more than one application domain. The most general aspect classes are Listable, SimpleTextEditable, Indexable etc. which represent just the most basic ways we organize our materials in every day work.

Every aspect class represents a specific view on a material class which offers the corresponding aspect. Thus, aspect classes are usually connected with their material classes using multiple inheritance.

Smalltalk, however, uses only single inheritance. Since we did not want to use a proprietary multiple inheritance solution we had to look out for another way of dealing with multiple views on a single material.

We finally decided to use protocol mixin to merge aspect classes into material class interfaces and to solve handling and maintenance issues by tool support.

Protocol mixin means that we still design our regular aspect classes and implement them as fully abstract classes. Thus, they define only a particular protocol but no implementation, which is in line with the original intention behind the aspect classes. A material implements the abstract state and functionality offered by an aspect class through an implementation state and based on its functionality.

Smalltalk offers no support for protocol mixin. We had to simulate it by copying the protocol defined by an aspect class into a specific material class. Doing this manually, however, would be cumbersome and error prone. Therefore we developed the Aspect Browser. An aspect class usually contains only abstract methods. The implementation is to be completed in the material class. The Aspect Browser helps to maintain and develop aspect classes and to copy their protocol definitions into material classes. It also offers a change history so that the method implementations are not lost if an aspect is removed from a material and inserted again.

So far using the Aspect Browser has been promising, however, in parts its handling has to be enhanced.

6 Conclusions

We have reviewed the Tools and Materials Metaphor and presented some of the important aspects of a Smalltalk framework to support application design based on this approach. The issues discussed include framework infrastructure support, tool construction (simple and complex tools, tool reuse) and tool coupling with materials (aspect classes, protocol mixin and tool support for multiple inheritance). These issues are covered in more depth in a technical report [RS95].

Issues that have not been discussed and implemented but have to be addressed in the future include connecting to legacy systems as well as new databases and other kinds of services. We have to introduce support for material administration on both a conceptual as well as concrete design and implementation level.

The framework currently consists of 75 classes with 800 methods. It is used as a starting point for an application in the banking domain. However, we expect that this first version will evolve and finally dissolve in the emerging overall framework of the developing application.

Bibliography

BCS92 Reinhard Budde, Marie-Luise Christ-Neumann and Karl-Heinz Sylla. "Tools And Materials: An Analysis and Design Metaphor." Tools-7, *Conference Proceedings*. Prentice-Hall, 1992. 135-146.

BGZ95 Ute Bürkle, Guido Gryczan and Heinz Züllighoven. “Object-Oriented System Development in a Banking Project: Methodology, Experiences, and Conclusions.” *Human Computer Interaction* 10, 2&3 (1995): 293-336.

GHJV95 Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. *Design Patterns: Elements of Reusable Design*. Addison-Wesley, 1995.

IBM94 IBM. *IBM Smalltalk Programmers Reference.* IBM, 1994.

KGZ92 Klaus Kilbert, Guido Gryczan und Heinz Züllighoven. *Anwendungsorientierte Softwareentwick-lung*. Vieweg, 1993.

Par94 ParcPlace Systems, *VisualWorks User’s Guide*, Sunnyvale 1994.

Rie95 Dirk Riehle. "How and Why to Encapsulate Class Trees." OOPSLA ’95, *Conference Proceedings*. To appear.

RS95 Dirk Riehle and Martin Schnyder. *Design and Implementation of a Smalltalk Framework for the Tools and Materials Metaphor.* UBILAB Technical Report 95.7.1. Union Bank of Switzerland, 1995.

RZ95 Dirk Riehle and Heinz Züllighoven. "A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor." *Pattern Languages of Program Design.* Addison-Wesley, 1995. 9-42.