ACTA UNIVERSITATIS APULENSIS No 11/2006

Proceedings of the *International Conference on Theory and Application of Mathematics and Informatics ICTAMI 2005* - Alba Iulia, Romania

DESIGNING A TASK ANALYSIS TOOL USING JHOTDRAW AND DESIGN PATTERNS

Grigoreta Sofia Moldovan, Adriana-Mihaela Tarţa

Abstract. Reusability is a desired characteristic of any software library, as it can reduce the time to develop a new software system and it can increase productivity. Frameworks and design patterns support reusability in two different ways: the former provides a skeleton for the new software (design and implementation), so the developers do not have to start from scratch and the latter provides solutions to recurrent problems, reducing the time needed to solve problems that might appear. GTATool is a task analysis tool prototype we are developing that support designers to build usable software systems. Usability is based on functionality, interaction between the users and the system, and presentation. All these aspects are treated carefully in the task analysis based methods like Groupware Task Analysis (GTA). The GTATool supports the design of software systems based on the GTA method. JHotDraw is a two-dimensional graphics framework for structured drawing editors. As the GTATool should support graphical task tree editing, it seems that JHotDraw is a good solution to start from. It has been developed using design patterns so it has all the advantages the design patterns have. Other design patterns like Composite and Singleton are used again to solve problems in the model layer of the tool. In this paper we present our approach in designing a GTA tool using JHotDraw and design patterns. The advantages and disadvantages we have observed throughout the design process are also presented.

2000 *Mathematics Subject Classification*: 68Nxx Software 68N19 Other programming techniques (object-oriented, sequential, concurrent, automatic, etc.)

1. Introduction

Nowadays, the Human Computer Interaction domain has gained the industry trust when developing software systems, focusing the design process around users' tasks and needs. Although the theoretical base is quite advanced, the free tools needed to support and help developers are lacking. The task analysis tool we are developing, called GTATool, has as theoretical base the Groupware Task Analysis method (GTA) [10]. GTA splits the design process in three steps: the analysis of the current situation (Task Model 1), the design of a new situation for the task (Task Model 2) and the design of a user virtual machine. GTA is developed based on an ontology where work is a very important concept. In the design process the work structure has a very important place because the interaction between the user and the system should be based on the natural order of subtasks or, if possible, should improve the task performance. Work structure has been described in different ways(more or less formal, textual or graphical). The best suited representation for work structure seems to be tree-like structure, because work structure is a hierarchical one where a task is performed accomplishing many subtasks and so on. That is why, a task analysis tools should mainly support the drawing and manipulation of task trees (adding children, copying nodes or subtrees, cutting or deleting nodes/subtrees, updating the tree properties). Although the user (in this case the software designer) works with a graphical representation, which is easier to understand, the tool should support the use of multiple representations (at least one formal representation for the work structure) that allows the validation of task models. The tool should keep consistency between representations.

For the graphical representation of a work structure we thought of using an exiting framework for editing task trees. Today there are some frameworks that can be used JFace [4], Graphical Editing Framework(GEF)[2] and JHotDraw [5]. We have decided to use JHotDraw as the last two are mostly used for rich GUIs and GEF is available only for the Eclipse platform.

Description of JHotDraw

JHotDraw is an application framework that can be used for developing custom-made drawing editor applications. Each application is targeted at a specific domain and reflects the domain's semantics by providing specific figure types

and by observing their relationships and constraints [8]. JHotDraw demonstrates frameworks' power and usefulness within their application domain [6].

The framework has been developed as a "design exercise" but it is already quite powerful. Its design relies heavily on some well-known design patterns [8].

Typical Development Process Using JHotDraw

In [6] a typical development process is described while using JHotDraw. The following is a list of recurring activities involved with developing an application with JHotDraw. The activities focus on integrating JHotDraw into an application and working together with the application model [6].

- **Create your own graphical figures and symbols for your application.** New applications usually require new graphical figures and symbols. Using JHotDraw they can be easily obtained by subclassing *AbstractFigure* or *CompositeFigure* and overriding the *draw()* method.
- **Develop your own tools to create and to manipulate figures according to application requirements.** The new figures require new tools to create them. To define the tools you can start from the existing ones such as: *CreationTool*, *ConnectionTool*, *SelectionTool*, and *TextTool*.
- **Create and integrate the actual GUI into your application.** To create and to integrate the GUI into the application you can subclass *DrawApplication* or *MDI_DrawApplication* for several internal frames or *DrawApplet*. They already contain methods for creating the menu and the tools. To change the default appearance you just have to override these methods.
- **Compile the application.** The last step is compiling the application using this framework.

Some Design Patterns used in JHotDraw

JHotDraw uses many of the design patterns presented in [1]. We only present here some of the design patterns it uses.

- **The Model-View-Controller paradigm.** JHotDraw is based on the Model-View-Controller (MVC) paradigm [1], which separates application

logic from user interface dependencies. JHotDraw is a view, and partly a controller. The developers are responsable for designing and implementing the model part and the interactions between the controller and the model.

- **The Composite design pattern.** The Composite pattern is used for composed figures (CompositeFigure). If, for instance, we need a figure that displays a Rectangle and a Text we can easily construct it by subclassing the *CompositeFigure* class.
- **The Strategy design pattern.** A *CompositeFigure* does not know how to draw itself, it does not know how to layout its components. That is why, this responsability is delegated to another class that subclasses *FigureLayoutStrategy*, and that contains the logic of walking through all child elements of the *CompositeFigure* and arranges those elements.
- **The State design pattern.** When you click a *CompositeFigure*, the appropriate child of the figure must handle the mouse click. To make sure that the right figure deals with the mouse click the State pattern is used.
- **The Template method.** When a subclass of *LineConnectionFigure* is created, the symbols at the end of the line might differ. That is why *LineConnectionFigure* provides *connectEnd()* and *disconnectEnd()* as Template methods.
- **The Factory method.** This kind of method is used extensively in JHotDraw, especially when creating user interface components such as menus and tools. Many Factory methods can be found in the *DrawApplication* class and have names like *createTools()* and *createMenus()*, which in turn call *createFileMenu()*, *createEditMenu()*, and so forth. If you want to change the GUI appearance you just have to override the appropriate Factory method.
- **The Prototype design pattern.** Each tool is initialized with an instance of the figure it is meant to create. Every creation tool in JHotDraw uses the original figure instance to create duplicate instances. The basic *clone()* mechanism is defined in *AbstractFigure*, where an instance copy is created using Java's serialization.

Figure **??** presents the overall architecture of JHotDraw.

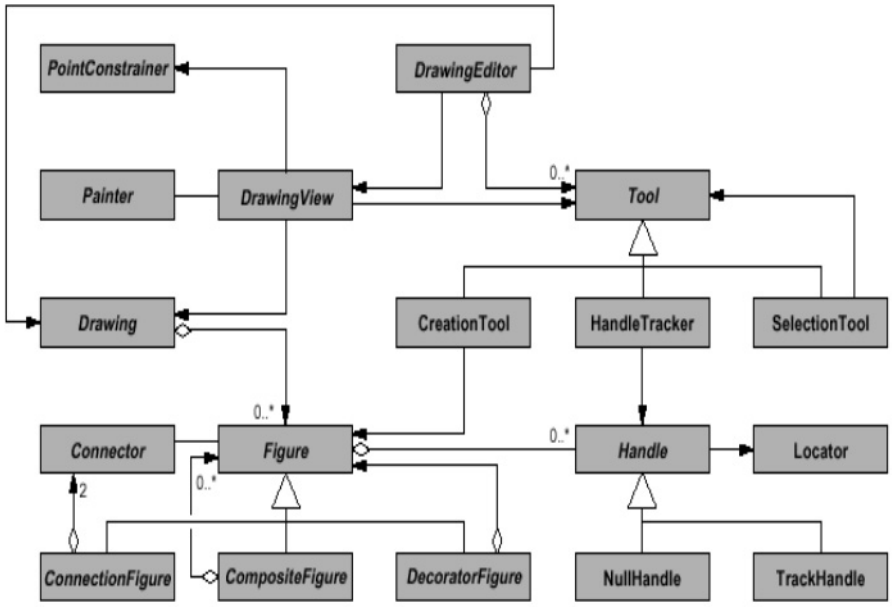

Figure 1: JHotDraw Architecture

2. Design of the GTATool

Designing the Task Tree Editor

A task analysis tool should support a various number of functionalities and should manipulate different types of information (audio, video, multimedia files, graphic or textual descriptions). In this article we will focus on the functionality related to the description of work structure. As we have mentioned in one of the previous sections, work structure is represented using task trees. In the following section we will give details about the types of tasks that can appear in a task tree, the constraints in the tree structure and the representation we have chosen for our tool.

Defining the Symbols and Figures

In order to create the task tree, the tool user can use four types of tasks: abstract, application, interaction, and user[7]. An application task reffers to those tasks that are performed by the system alone (sending a message on a

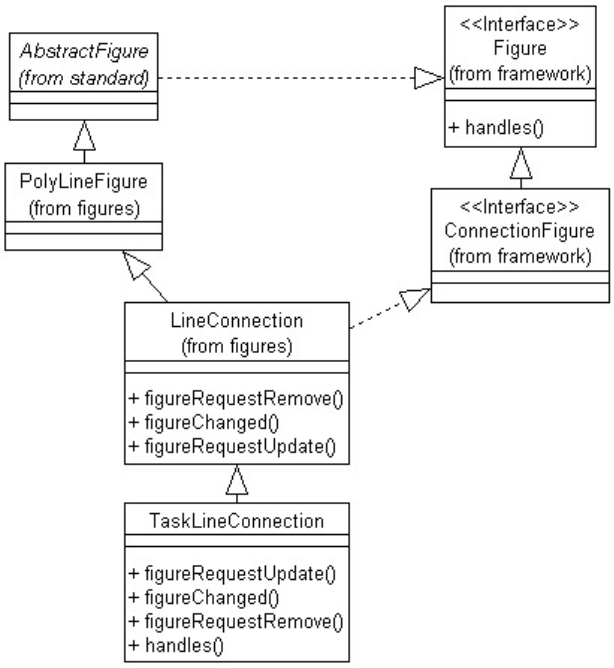

Figure 2: The TaskLineConnection class hierarchy

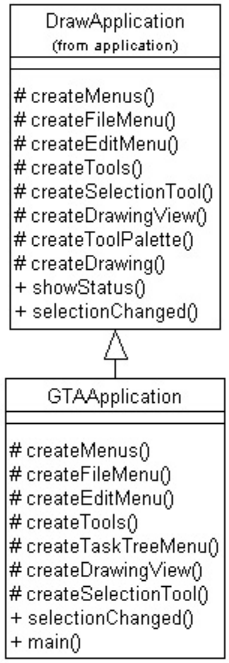

Figure 3: The GTATool class hierarchy

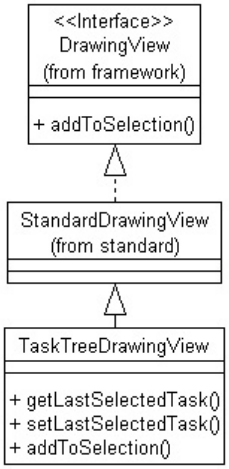

Figure 4: The TaskTreeDrawingView class hierarchy

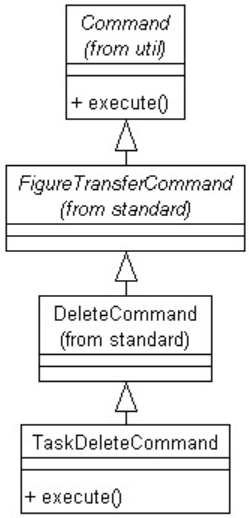

Figure 5: The class hierarchy for TaskDeleteCommand

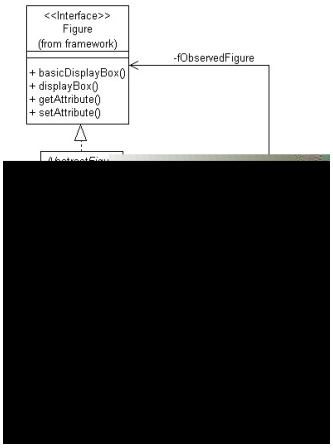

Figure 6: The TaskFigure class hierarchy

Figure 7: The TaskCreationTool class hierarchy

network, saving a file); a user task is a task performed by the user involving a physical activity or a mental activity; an interaction task is a task performed by the user in collaboration with the system (editing a text box), and an abstract task is one that is too complex to be classified in one of the above mentioned task types. An abstract task will always be the root of a task tree and will have at least two children (subtasks). There are several constraints regarding the structure of task trees: the root should always be an abstract task, an abstract task can only be the child of another abstract task, a user task or an application task can only have user, respectively application tasks as children. An abstract task cannot be the leaf of a tree. Each type of task has a different associated symbol, but they all have a name that must appear under the task's symbol. That is why we have decided to subclass the *CompositeFigure* in order to create a *TaskFigure*. Our *TaskFigure* uses two other figures: a *ImageFigure* for the task type and a *TextFigure* for the task name (Figure **??**). The *ImageFigure* is given to the *TaskFigure* at creation time.

The tool must create a task tree, so we should be able to connect two tasks. For that we have defined a *TaskLineConnection* that does not have any symbols at its two edges (Figure **??**). Before creating the actual connection, it first verifies whether the two tasks can be connected(based on the tasks' types and on the requirements constraints).

Defining the Creation Tool

As the connection between two tasks is done automatically at creation time, we do not need to define a creation tool for *TaskLineConnection*.

Although we have four task types, we only need one creation tool, because of the creation mechanism used by JHotDraw. Our *TaskCreationTool* subsclasses the *CreationTool* and overrides the methods shown in Figure **??**. When we create a new task, we have to automatically connect it to its parent. That is why we introduced here the creation of *TaskLineConnection*. *TaskLineConnection* must know its two edges, so we should memorize the last selected *TaskFigure*, which will be the parent of the newly created *TaskFigure*. As in the JHotDraw framework, the *DrawingView* is responsible for knowing which figures have been selected, we have defined our *TaskTreeDrawingView* (Figure **??**) to keep track of the last selected *TaskFigure*. For that, all we have to do is override the *addToSelection()* method.

Defining the Actual GUI

As the first prototype of our task analysis tool does not use more internal frame, in order to create the GUI we have subclassed the *DrawApplication* class. To define the particular appearance we want for our GTATool, we just have to override some factory methods like *createTools()*, *createMenus()* (Figure **??**).

Designing Other Functionalities

In a drawing editor the user is usually able to delete some parts of the drawing. This is also possible in the task tree editor. The tool's user can delete a task or a subtree, or even the entire task tree. Designing this functionality is very easy as the framework already contains a similar functionality which was designed using the Command design pattern [1]. All we have to do is subclass *DeleteCommand* and override the *execute()* method (Figure **??**).

Designing the Model

Any task analysis tool should be able to process and save the task tree in different formats for further analysis. So, we need to somehow represent the task tree internally. As a task may have other tasks as children, we have decided to use again the Composite design pattern.

One of the formats in which a task tree can be saved is the XML format, which corresponds to the DTD we have defined in [9]. In order to preserve the tree-like structure for further use and analysis, we have defined two attributes of type IDREF for each task in the task tree: one of them represents the task ID and the other represents the parent ID. As the values for these attributes must be unique for each task, we used the Singleton design pattern to ensure their uniqueness.

3. Conclusions and Further Work

Developing new applications using JHotDraw framework is very easy and it takes little time. The only inconvenience we have observed is the lack of a good documentation. Luckily, the source code is freely available, so we could browse it to gain deeper understanding of the frameowrk. Using this framework we have very quickly developed our first prototype of the GTATool. All we had to do is subclass some classes and override some methods, and the prototype was ready. Another benefit from using this framework is that our application has good architecture, that allows further changing and evolving.

One functionality we desired of our tool is to automatically obtain the GUI from a task tree after adding some specific details related to task types (like monitoring, editing, notification, etc). For that we have to introduce temporal operators, such as LOTOS [3], in our task tree editor. The introduction of temporal operators can be easily done using JHotDraw. We just need to define their symbols, to define creation tools for them, and to add the tools to the GUI.

References

[1] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object Oriented Software*, Addison-Wesley, 1995.

[2] GEF: http://www.eclipse.org/gef/.

[3] ISO/IEC. Information Processing Systems - Open Systems Interconnection, *LOTOS, a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, IS 8807, 1989;

[4] JFace: http://www.eclipse.org/ .

[5] JHotDraw: http://www.jhotdraw.org .

[6] Kaiser, W.,*Become a programming Picasso with JHotDraw*, http://www.javaworld.com/javaworld/jw-02-2001/jw-0216-jhotdraw.html, February, 2001.

[7] Paternò, F. and Mancini, C. and Meniconi, S.,*ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models*, INTERACT '97: Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction, 1997,ISBN 0-412-80950-8,pp. 362-369, Chapman & Hall, Ltd.

[8] Riehle, D., *Framework Design. A Role Model Approach*, ETH Zurich, PhD. Thesis, 2000.

[9] Tarta, A. M., Moldovan, G. S. *GTATool- A Task Analysis Tool*, Proceedings of the Symposium "Colocviul Academic Clujean de Informatica", Cluj-Napoca, June 2005, to appear.

[10] van Welie, M.,*Task-based User Interface Design*, Vrije Universiteit Amsterdam, PhD. Thesis, 2001;

Grigoreta Sofia Moldovan
Department of Computer Science
Babeş-Bolyai University
Address: Str. M. Kogălniceanu, Nr. 1, Cluj-Napoca, Romania
email:*grigo@cs.ubbcluj.ro*

Adriana-Mihaela Tarţa
Department of Computer Science
Babeş-Bolyai University
Address: Str. M. Kogălniceanu, Nr. 1, Cluj-Napoca, Romania
email:*adriana@cs.ubbcluj.ro*