HotDoc
A Framework for Compound Documents

Jürgen Buchner
Darmstadt University of Technology — Programming Languages and Compiler

This paper presents HotDoc, a framework for the development of editors for compound documents. HotDoc allows the construction of flexible documents consisting of dynamic parts. HotDoc introduces a new type of document. A document is not only a static sequence of text, but an interface to small applications (parts). Programmers can easily implement new parts by using abstract classes of the framework. HotDoc is implemented in VisualWorks Smalltalk.

Categories and Subject Descriptors: D.1.5 [**Software**]: Programming Techniques—*Object–oriented Programming* I.7.2 [**Computing Methodologies**]: Text Processing—*Document Preparation*

General Terms: Design, languages

Additional Key Words and Phrases: Frameworks, Compound documents, Multimedia, Smalltalk

1. INTRODUCTION

With the development of WWW, hypertext help systems etc. the domain of document processing turns into one of the most important fields in computing technology . Originally text processing systems were simply used as a replacement for the typewriter with the additional functionalities easy correction and reuse of existing documents.

With the increasing power of computer systems it was possible to integrate more functions in the text processors. Not only text, but also images, tables, mathematical formulae etc. can be part of documents. Such documents are called *compound documents*, they consist of *parts*. Each part has a specific *document type* like text,

Address: Darmstadt University of Technology (TUD)
FB 20 – FG PÜ
Alexanderstr. 10
D–64283 Darmstadt (Germany)

E-Mail: buchner@pu.informatik.tu-darmstadt.de
Phone: +49-6151-16-3610
Fax: +49-6151-16-6648

graphic, sound etc. Tools are needed for constructing such compound documents.

This paper presents HotDoc [Hauck 1996; Buchner 1997; Buchner et al. 1997], a Smalltalk framework [Fajad and D.Schmidt 1997] for the construction of compound documents.

2. THE DOCUMENT MODEL OF HOTDOC

In most applications, a document is a linear sequence of text. Sometimes pictures, tables etc. are included in the text. Such a document is static and can be printed on paper.

A HotDoc document is like a playground: Authors can place parts on such a document. Parts are like windows to applications. Examples are text editors, spreadsheets or video players. By combining different parts in one document, new functionality can be created.

Such documents can be used as *document–based user interfaces*, like in the Hot-Simple system [Buchner et al. 1997]. By inserting parts, the user creates an interface specific to his or her problem. There is a seamless transition between *documents* and *user interfaces* in HotDoc.

Because of its dynamic nature, the main purpose of such a document is not to be printed on paper. But a HotDoc document is well suited to be transmitted electronically from one user to another.

3. DESCRIPTION OF HOTDOC — USER'S VIEW

When HotDoc is started, the user interface shown in figure 1 is displayed. HotDoc uses an interface similar to many standard applications in the office domain.

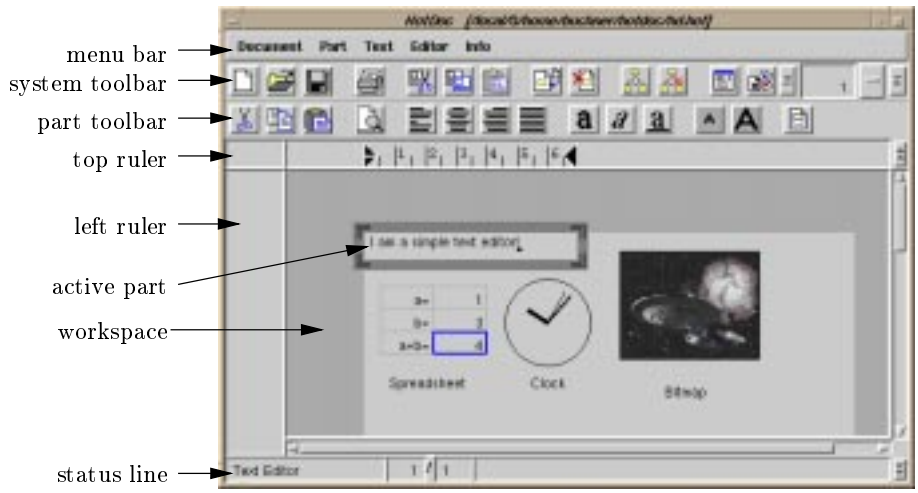

Fig. 1. The HotDoc user interface

A new, empty document consists of a *workspace* only. The workspace is a special part, the root of a HotDoc document.

The user constructs his or her document by inserting new parts into the workspace.

Parts can be nested, they can contain other parts. The part developer decides, if a part accepts the insertion of other parts.

Part Insertion

To insert a part, the user opens the *Insert Part* dialog and selects the part's document type from a selection list. Information about the part such as the author and a short description is displayed.

In the other sections of the *Insert Part* dialog, the user can specify the placement of the part inside of its parent and the strategy to be used to place parts inside the new part[1].

Active Part

When a part is inserted, it is automatically activated. The activated part of a document is the part the user is currently working with. This part is marked with a thick grey frame. In figure 1 a text editor part is activated. By clicking and dragging this frame, the part can be resized and repositioned. The user can activate another part by clicking on it.

The user interface of an active part is merged with the interface of the HotDoc system: The part's menus are included in the menu bar, the part's tool bar is displayed, the part's rulers are activated and the part's name is shown in the status line. This merging technique has the advantage, that the user interface basically stays the same. Additionally, written guidelines for the part designer achieve a uniform behavior.

All keyboard events are processed by the active part (except for some system shortcuts).

Document structure

Since parts can be inserted in other parts, a HotDoc document is a tree of parts. This tree can be visualized by the system. The *document structure* dialog is shown in figure 2.

Fig. 2. The "document structure" dialog

[1] This feature is called *layout policy* and will be explained later.

In HotDoc the terms *parent* and *child* are used to describe the relationship between parts analogously to the relationship between nodes in a tree.

There is no limit in the depth in which parts can be nested. Furthermore, all parts in the tree are displayed and can be activated and edited in place.

Model Linking

Parts can share a common data model. This is done by *linking* two or more parts together. The types of the linked parts may differ, but they must be *link compatible*.

E.g., a business graphics part can be linked with a spreadsheet part. Whenever the data in the spreadsheet is changed, the business graphics is updated automatically.

Layout Policies

When one part is inserted into another, the placement of the parts is crucial. On one hand, the user should not need to place each part manually, on the other hand, a great flexibility of placement is necessary.

HotDoc solves this requirement with so–called *layout policies*. A layout policy is assigned to a part and controls the placement of its children.

When the user moves a part with the mouse, the movement is constrained by the layout policy. After a movement or the insertion of a new part, or if the parent is resized, the layout policy rearranges all children if necessary.

4. IMPLEMENTATION OF HOTDOC

The implementation of the HotDoc framework consists of several components:

—A class library containing abstract and concrete classes which are used by the HotDoc system itself and are also supporting the design of new parts.

—The HotDoc frame window (figure 1). Through this window, the user interacts with HotDoc and the parts.

—A standard part library. This is a set of standard parts which are useful for the construction of documents. This library includes a text editor, a bitmap viewer and some other parts.

HotDoc is implemented in VisualWorks Smalltalk [ParcPlace Digitalk 1995] and uses the *Model–View–Controller paradigm* (MVC–paradigm, MVC, [Krasner and Pope 1989]) of Smalltalk. The class library of VisualWorks Smalltalks includes a mechanism to crate nested views. HotDoc uses and extends this mechanism to implement parts. The HotDoc framework is a extension of Smalltalk's MVC–framework. Most HotDoc classes are subclasses of predefined Smalltalk classes; they don't form a class hierarchy of its own. Figure 3 shows some important HotDoc classes.

To implement a new part, the programmer can use and extend classes of the HotDoc framework.

Until now, several parts have already been developed: A spreadsheet, a business graphics part, a vector graphics editor and some other smaller parts.

We currently work on publishing documents on the WWW with on–the–fly conversion to HTML, using HotDoc as a tool for CSCW and the development of a visual scripting language for inter–part communication.

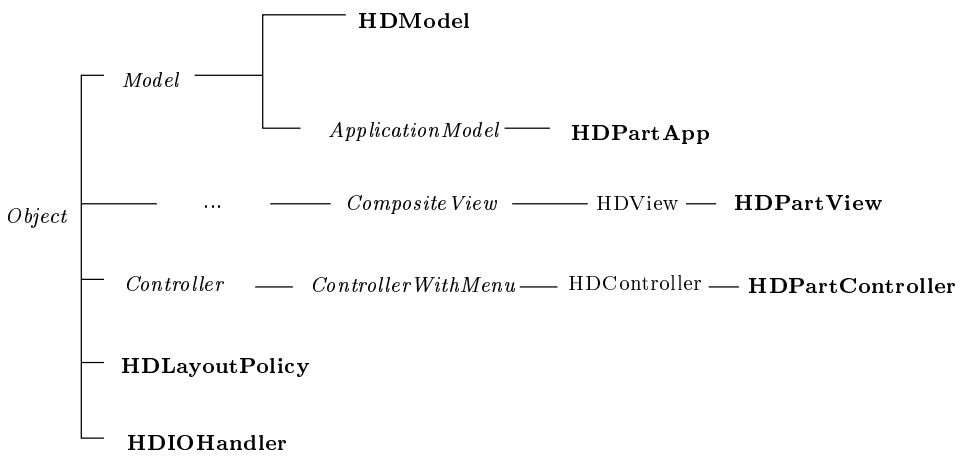

Fig. 3. Some important classes from the HotDoc class hierarchy (VisualWorks classes are printed in *italics*, classes relevant for part designers are printed **bold**)

5. CONCLUSION

HotDoc is a framework, which can be used as a stand–alone system for compound documents and as a development platform for many other applications.

For the programmer, using HotDoc has, among others, the following benefits:

—Persistence of the parts and documents is handled by the framework. When the user stores a document, the data of all parts are saved automatically. When a document is loaded, all parts are recreated according to the stored data.

—Many aspects of user interface programming are handled by the framework. It is responsible for moving and resizing, cutting and pasting, scrolling etc. The programmer can concentrate on the implementation of the application logic.

—HotDoc is implemented in VisualWorks. The part designer has all advantages of this object–oriented programming system, like automatic garbage collection, the huge class library and the development environment.

And the user of a HotDoc–based application gains some benefits:

—Since many aspects of the application's interface are handled by the framework, all HotDoc parts behave similar.

—The user is able to combine different parts in one document. E.g., a user may combine a product description inside a text–editor part and the calculation inside a spreadsheet part.

—The user is assisted by layout policies in the task of arranging parts.

Unlike many other systems (e.g. OpenDoc [Nelson 1995], Microsoft OLE [Microsoft Corporation 1992], Andrew Toolkit [Plotkin et al. 1993] or OOE [Backlund 1997]), HotDoc is designed for easy programming of new parts. By using an object–oriented framework and Smalltalk, it is "easy" to create new parts for the programmer *and* it is"easy" to use for the end–user.

REFERENCES

Backlund, B. E. 1997. OOE — A compound document framework. *SigCHI Bulletin 29*, 1 (Jan.), 68–75.

Buchner, J. 1997. HotDoc — Ein Anwendungsrahmen zum Aufbau von Dokumenten aus Bausteinen. In *Internationales Wissenschaftliches Kolloquium*, Volume 1 (1997), pp. 30–35. Technische Universität Ilmenau.

Buchner, J., Fehnl, T., and Kunstmann, T. 1997. HotDoc — A flexible framework for spatial composition. In *IEEE Symposium on Visual Languages* (1997).

Fajad, M. and D.Schmidt. 1997. Object–oriented application frameworks. *Communications of the ACM 40*, 10.

Hauck, M. 1996. Entwurf und Implementierung einer Klassenbibliothek zum Realisieren zusammengesetzter Dokumentstrukturen. Master's thesis, TH Darmstadt, FB Informatik.

Krasner, G. E. and Pope, S. 1989. A cookbook for using the model–view–controller user interface paradigm in Smalltalk–80. *Journal of Object–oriented Programming 1*, 3, 26.

Microsoft Corporation. 1992. Microsoft Backgrounder: Object linking and embedding 2.0. Technical report (Nov.).

Nelson, C. 1995. OpenDoc and its architecture. *AIXpert*.

ParcPlace Digitalk. 1995. *VisualWorks Cookbook*. ParcPlace Digitalk.

Plotkin, A., Hansen, W. J., et al. 1993. A user's guide to the Andrew User Interface System. Technical report, Carnegie Mellon University.