Reliable Component Development for End Users

Ludger Martin
Peter–Weyer–Straße 37, 55129 Mainz, Germany
lumartin@NotaMusica.com

Abstract: In recent years, the opinion existed that end user developers create more applications than professional developers. Therefore special development environments are needed to support the end user developer as best as possible.

In this paper, I present parts of the HOTAGENT component development environment especially arranged for end users. The focus of the investigation is on developing reliable components. Special tools are presented to support the end user developer. Visual effects show the end user developer the quality of a component or the need of additional tests etc.

1 Introduction

Creating reliable software is still a contemporary issue. Many companies believe that they don't need extensive tests and maintenance strategies. But not only professional software developer are compelled to test. Especially end user developers need to be stimulated to test there programs. Development environments and their languages need to be easy but flexible. To provide this component development environments are especially suited this way.

In this paper tools will be presented to support the end user developer (EUD) to develop reliable components. They visualize the EUD the reliability of components and the need of specifying more tests or assertions for a component. They are part of the HOTAGENT development environment [MM03, Mar03]. In the next section we discuss the basic concepts of end user development, components, and testing. The tools to support the EUD during development are presented in Section 3. The paper concludes with a summary and a discussion of future work in Section 4.

2 Basic Concepts

To discuss the development of reliable components for end user, first it is necessary to present concepts of end user development. Later on, we define the term component and explain strategies in testing.

Nowadays end user development is the most common form of programming. Mørch *et al.* [MSW+04] define end user development as customizing of applications by the end user.

The end user has expert and technological knowledge but only low or no knowledge in programming at all.

End user development is done by customizing applications. To facilitate this, parts to build an application must be under-designed at design time. Thereby the end user can create special applications. He can make complex customizations or extensions.

Fischer *et al.* [FGY+04] point out a conflict between complexity and capability. To allow to solve a huge amount of tasks the complexity of the development system increases and it is harder to be handled by the end user. In my opinion the use of components can solve this conflict.

Szyperski [Szy98] defines: "A *component* is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

According to Lüer and Rosenblum [LR01], a component should have a public and a private part. The *interface* is the sole public part of the component. All other classes of the component are private. The interfaces ensure the *communication* between all the components. In our component model, every component provides *entrances* and *exits*. Communication between components is possible only through these public entrances and exits. An entrance can receive data and produce a result depending on the component. An exit is activated if a component has to show any change by providing the appropriate data. To compose components, exits are connected with entrances.

Mørch *et al.* [MSW+04] explain three phases of component development for end users. In the first phase *customization* the end user modifies parameters of a component. *Extension* is done by creating new components by programming with source code. *Integration* is the phase of creating or modifying applications using components.

Now testing will be discussed. McCarthy [McC97] defines an unit test as: "testing a function, module, or object in isolation from the rest of the program." A unit is normally the smallest item of programming. In component technology, the smallest unit is a component. It is not useful or possible to break down the components and test the encapsulated classes. Tests for component software need to regard components as black boxes.

Since we understand a component as a block with entrances and exits, a message or data can be sent into the entrances and can be tested against the data coming from the exits. The tester must define the data of the individual exits before checking the answers. The combination of data sent into entrances and data expected at exits is called test fixture. If an exit has a wrong value or is not activated, a failure has occurred.

Beside testing it is also possible to use Design by Contract. Assertions have to prove the correct functionality of a component. These assertions are checked at run time. Using Verified Design by Contract formal verification during design time is done. This can help the EUD during assembling components. So the EUD can correct the problem at the moment of programming.

3 Reliable Component Development

Fischer *et al.* [FGY+04] point out that EUD need to be well motivated. Therefore HOTAGENT provides tools to support the user by creating components, by creating applications, by testing, and by analysis. The next paragraphs describe how to create components, how to specify assertions for the components and how to test components.

If the EUD has determined that a new component needs to be built to implement the application, HOTAGENT COMPONENT [MM03, Mar03] should be used. To design a new component, entrances and exits can be created to specify the interface. In addition to specifying descriptions (self documentation) for the interface, it is required to indicate an assertion for every entrance and exit. If the EUD specifies an assertion the color of the entrance or exist is changed.

New Components are implemented using existing components. Having selected the interior components it is necessary to connect these components among each other and to the interface ports (entrances and exits) of the new composed component. It is possible to connect one exit to several entrances and several exits to one entrance.

To support especially the EUD HOTAGENT COMPONENT shows the quality of the new constructed component in the toolbar. The quality is measured by assertions specified for the interface of the new component and by tests specified for the new component (see below). A second indicator is the average quality of all interior used components.

During connecting components *Verified Design by Contract* is used. The postcondition and precondition of the exits and entrances are checked against each other. If they don't fit, the EUD will be notified. During run time the assertions are also checked (*Design by Contract*). Because of this the EUD is always informed where to tackle a problem.

Apart from displaying quality criteria, the editor offers the possibility to label and color connections to support *secondary notation*. It also offers a number of display filters (*level of detail* or *separation of concern*) to reduce the amount of information displayed on the workspace.

Using the HOTAGENT COMPONENT editor the EUD is well supported by creating and maintaining components. By specifying assertions he can increase the reliability. But this is not enough. The EUD has to test it's component which is described below.

The HOTAGENT TEST tool [MM03, Mar03] is used to specify test fixtures for components and composed components. Every test fixture can be created according to the following scheme: create all components to test, connect the components, send data to a component entrance, and define an assertion for a component exit.

HOTAGENT TEST can be used in the same way as HOTAGENT COMPONENT to place and connect components. Data to send to a component entrance can be created using a special data component. Similarly, a test component can be used to complete a test fixture. These two components can interpret any Smalltalk code to define a data set or to specify the test block. If the test block is evaluated as true, the test is positive. When the test is executed, the color of the test block component changes in relation to the result. HOTAGENT TEST is able to store a test fixture with every component or component program.

HOTAGENT REGRESSION is a tool to run regression tests. It is similar to the SUnit/JUnit test runner. It can run a specific test case or all test cases defined in a system.

4 Summary and Future Work

In this paper, I present some parts of HOTAGENT as a development environment suitable for end users. The main focus is on creating reliable components. The tools presented try to support the end user developer as best as possible by creating and maintaining reliable components.

At the moment not all tools of HOTAGENT are revised to be suitable for end user developers. Investigations are in process how to support end user developers in all development steps in a unified tool set with one consistent user interface and a common operational paradigm. An assessment of HOTAGENT for EUD has to be accomplished.

A hard question is how to know whether a component is reliable or not. Is it sufficient to specify assertions and test cases? Are there any conditions these assertions and test cases need to met. McCarthy [McC97] claims that it is always uncertain if a test suite is complete. He proposes to test until finding several failures. It is also necessary to investigate a different solution. Another interesting question is how much testing is needed for a composition of components if the components have already been tested in isolation.

Literatur

[FGY+04] G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe und N. Mehandjiev. Meta-design: a manifesto for end-user development. *Commun. ACM*, 47(9):33–37, 2004.

[LR01] Chris Lüer und David S. Rosenblum. Wren – An Environment for Component-Based Development. In *Proceedings of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Seiten 207 – 217, September 2001.

[Mar03] Ludger Martin. *Visuelles Komponieren und Testen von Komponenten am Beispiel von Agenten im elektronischen Handel*. Dissertation, Darmstadt University of Technology, 2003.

[McC97] Adrian McCarthy. Unit and Regression Testing. *Dr. Dobbs Journal*, Februar 1997.

[MM03] Ludger Martin und Johannes Martin. HotAgent: Round Trip Component Development. *Annals of Mathematics, Computing & Teleinformatics*, 1(1), 2003.

[MSW+04] Anders I. Mørch, Gunnar Stevens, Markus Won, Markus Klann, Yvonne Dittrich und Volker Wolf. Component-based technologies for end-user development. *Commun. ACM*, 47(9):59–62, 2004.

[Szy98] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998.