

A Human-Centric Tool for Component Design and Reuse in the UML *

E.E. Roubtsova

Technical University Eindhoven,
Den Dolech 2, 5600 MB The Netherlands,
E.Roubtsova@tue.nl

S.A. Roubtsov

VTT Electronics, Kaitovayla 1, P.O.Box 1100,
FIN-90571 Oulu, Finland,
ext-Serguei.Roubtsov@vtt.fi

Abstract

We present a tool for component system design in the UML. The tool supports an internal process model for graphical specifications. The model is used to guarantee consistency of different graphical views in the component specification. The model allows the tool to manage the graphical views composition, so that the composed specification inherits internal process models of components.

1. Introduction

Abstract diagrams of the Unified Modeling Language (UML) [3] help designers quickly understand semantics of specifications. However, if a designer constructs a precise specification in the UML and composes one specification from other specifications, he should keep in mind a lot of information: the correspondence of different diagrams; rules for composition of diagrams and links between specifications at different levels of abstraction. The UML itself defines no way for consistent design and reuse. Some of existing tools, such as Rational Rose, support the name space consistency within a single model, but there is no support for correct specification reuse.

In this paper, we present a Rational Rose Add-In which supports the consistent design of components and the specification reuse in a UML profile [1]. The design is driven by an internal process model built into the tool. The component specification reuse and the rules for composition of specifications are based on the inheritance of processes [5]. Constructing the internal model and diagrams corresponding to this model is the task of the tool. The reuse and extension of a component model mean in the tool the reuse and extension of both the internal model and the diagrams.

It is psychologically proven that an average human being can only fix his attention on maximum five to seven objects on a picture at one particular moment. That is why, our tool provides a specification methodology. If a designer

follows the tool methodology, then the tool fulfills some of human tasks: the reused diagrams are loaded on the screen to be changed according to the predefined inheritance rules, the inheritance of internal process models is checked by the tool, the bugs of the design can be found, the elements of the diagrams are drawn by the tool automatically using the information filled in the tool dialogs, the library of components is collected.

In Section 2 we briefly describe the component model used by the tool. Section 3 informally explains the inheritance relations on different diagram levels built into the tool. Section 4 overviews the design methodology provided by the tool. Section 5 contains some conclusions.

2. The Component Model Used by the Tool

A component specification in our tool is a process term. The process term of a component is specified by a designer in the UML component profile by an interface-role diagram (a variant of the class diagram [2, 1]) and a set of sequence diagrams. The set of action names of the process term is derived from the UML interface-role diagram (Fig.1b). Using all of these diagrams (Fig.1a,c) our tool constructs the process term (Fig.1d) [1] and the corresponding process graph.

Thus, the specification of a component in our tool contains three consistent parts: an interface-role diagram, a set of sequence diagrams and the process term being an internal model for all diagrams.

3. The Composition Techniques Built into the Tool

We have transformed the definition of the process inheritance [5] into the definitions of inheritance for interface-role diagrams and for sequence diagram sets. Those definitions have been built into the tool. Here we present an informal interpretation of the definitions which have been precisely described in [1].

To construct interface-role diagram IR_q of component-inheritor from parent interface-role diagrams IR_{p_i} , we extend parent interface-role diagrams by an interface-diagram IR_{new} (Fig.2). Some roles of IR_{new} inherit some roles of IR_{p_i} , $i = 1..n$. (If role r_1 inherits role r_2 , then we note

*The work of S.A. Roubtsov is supported by The European Economic Interest Grouping ERCIM (European Research Consortium for Informatics and Mathematics). His work is a part of VTT Electronics Agile project: <http://agile.vtt.fi>.

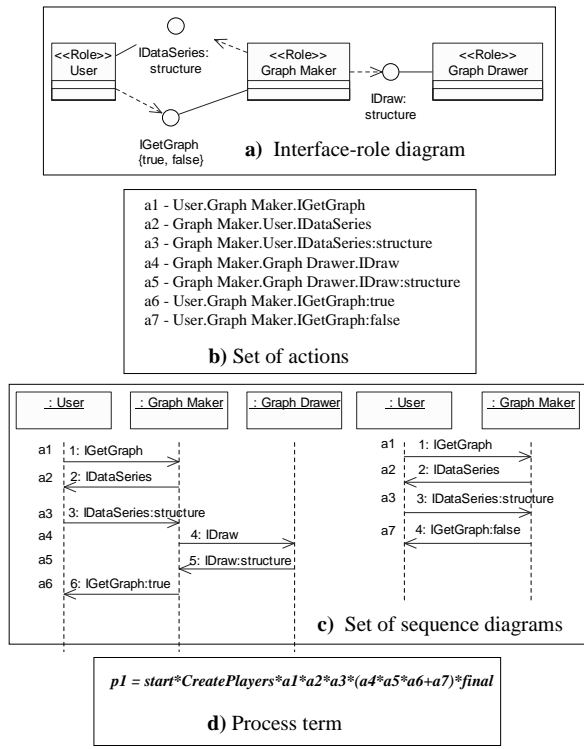
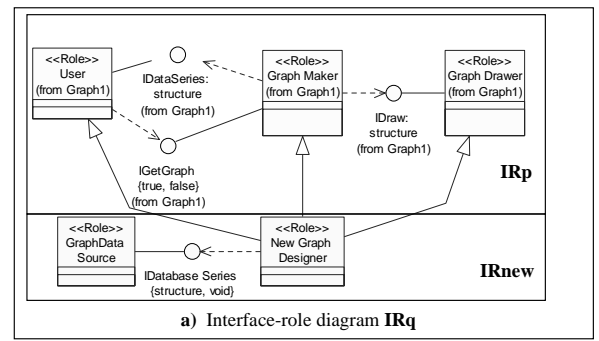


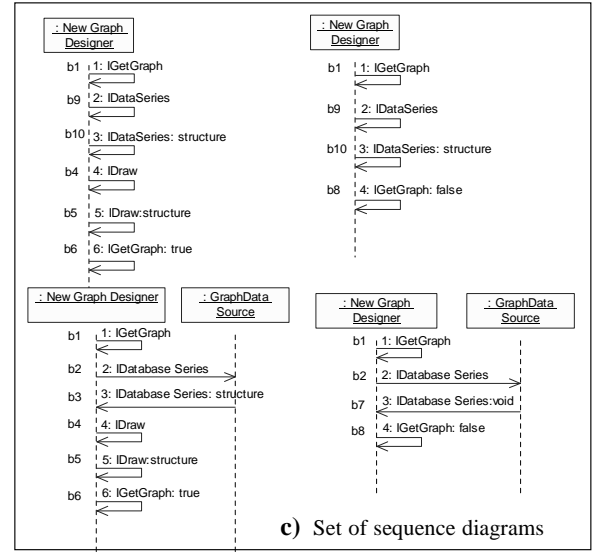
Figure 1: Component-parent

this as follows: $r_1 \rightarrow r_2$ [3]). The roles of interface-role diagram IR_{new} (Fig.2a) cannot require interfaces of parent roles from interface-role diagrams IR_{p_i} , and roles from IR_{p_i} cannot require interfaces of roles from IR_{new} . To reuse an interface provided by a parent role and required by another parent role, both parent roles should be inherited by roles of the interface-role diagram IR_{new} . This way the parent interface and the corresponding actions are duplicated in the interface-role diagram IR_q of the new component (e.g. a_1 and b_1 Fig.2).

Each new component has its own set of sequence diagrams (Fig.2c). To check that this set of sequence diagrams inherits parent processes, designers should compare the process term of the new component (Fig.2d) with the process term of each parent component (Fig.1d). However, those process terms are incomparable, because they have different sets of actions (Fig.1b and Fig.2b). To make the processes comparable, the tool derives duplicating functions from the interface-role diagram. A duplicating function relates the actions of a parent component and the actions of its inheritor. Our tool allows us to rename a parent process to make it comparable with the process of the inheritor (Fig.2e). After renaming, the parent process should be derived from the process of the new component in the process algebra with hiding and blocking functions [1]. We have formalized no-



- b) Set of actions**
- b1 - New Graph Designer.New Graph Designer.IGetGraph
 - b2 - New Graph Designer.GraphData Source.IDatabase Series
 - b3 - New Graph Designer.GraphData Source.IDatabase Series: structure
 - b4 - New Graph Designer.New Graph Designer.IDraw
 - b5 - New Graph Designer.New Graph Designer.IDraw:structure
 - b6 - New Graph Designer.New Graph Designer.IGetGraph: true
 - b7 - New Graph Designer.GraphData Source.IDatabase Series:void
 - b8 - New Graph Designer.New Graph Designer.IGetGraph: false
 - b9 - New Graph Designer.New Graph Designer.IDataSeries
 - b10 - New Graph Designer.New Graph Designer.IDataSeries: structure



d) Process term

$p2 = \text{start}^* \text{CreatePlayers}^* b1^* (b2^* b3^* (b4^* b5^* b6 + b7^* b8) + b9^* b10^* (b8 + b4^* b5^* b6))^* \text{final}$

e) Renamed parent process term

$p1i = \text{start}^* \text{CreatePlayers}^* b1^* b9^* b10^* (b8 + b4^* b5^* b6)^* \text{final}$

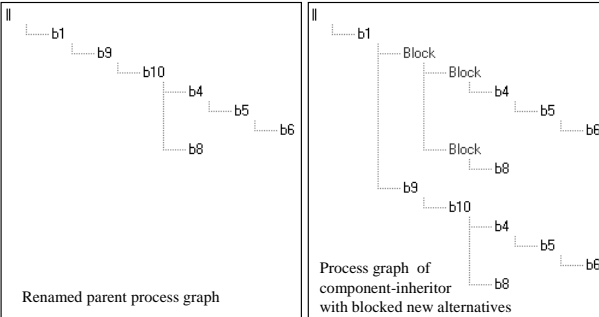


Figure 2: Component-inheritor

tions of correct inheritance and developed process rewriting and process-graph transformation rules to prove process inheritance (Fig.2f).

4. The Design Methodology Provided by the Tool

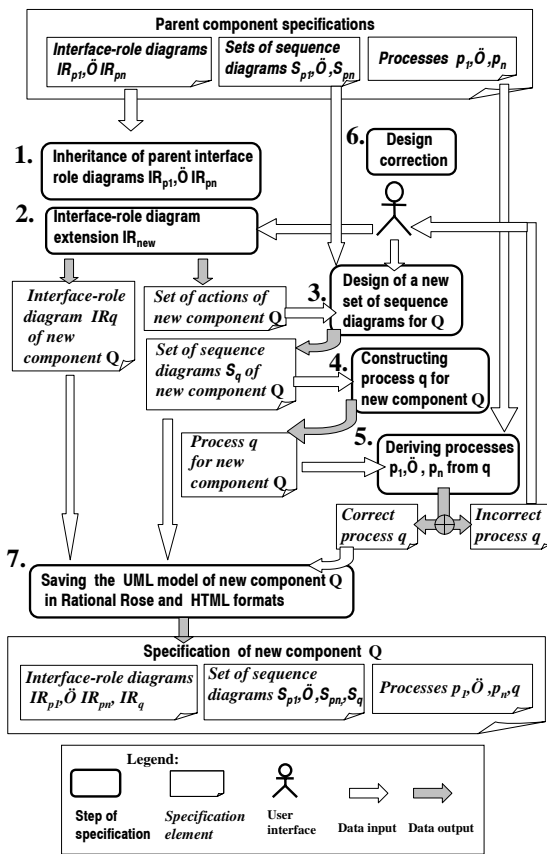


Figure 3: Methodology provided by the tool

Fig.3 shows the steps of the component system design in the tool:

1. A designer chooses parent components to inherit from. Interface-role diagrams of these components are drawn by the tool on the screen.
2. The designer extends the parent interface-role diagrams with new roles and interfaces using dialogs provided by the tool. The interface-role diagram of the new component is produced.
3. The designer draws a set of sequence diagrams using the set of actions derived by the tool from the interface-role diagram of the new component.
4. The tool constructs the process term corresponding to the UML specification of the new component.

5. With the help of the tool, the designer tries to derive processes of parent components from the process of the new component.

6. If the derivation is not successful, then the sequence diagrams that represent the unreachable behaviour patterns are indicated by the tool. The designer should correct the design of the new component.

7. If the derivation is successful, then the specification of the new component is saved in three different sets of files:

- a. A Rational Rose model (*.mdl'- file). This file provides an access to further component modifications.
- b. A set of Rational Rose category files with special extension '*.sui'. This set contains specifications of the component and all its predecessors.
- c. A set of documentation files. It includes the main HTML-file with the component specification and a set of graphic files in EMF format containing interface-role and sequence diagrams. The documentation is generated by the tool automatically. A designer can choose the contents of the documentation.

5. Conclusion

Our tool is a Rational Rose Add-In, which provides the inheritance of previously specified components, single and multiple, nested with any depth. We have carried out about ten case studies of different complexity with the help of the tool [4]. Our evaluations show that to build a correct model within our tool, designers spend about 5 to 7 times less time than within the conventional Rational Rose environment. The formal semantics of the UML built into the Rose Add-In helps prevent semantic bugs hidden in the behavioural inheritance of component specifications. The methodology provided by the tool supports reuse of graphical and internal process specifications, design correction and documentation.

References

- [1] E.E. Roubtsova, R. Kuiper. Process semantics for UML component specifications to assess inheritance. *Electronic Notes in Theoretical Computer Science*, 72,3,Editors P.Bottoni, M. Minas, 2003.
- [2] J. Cheesman, J. Daniels. *UML Components. A simple Process for Specifying Component-Based Software*. Addison-Wesley, 2001.
- [3] OMG. *Unified Modeling Language Specification v.1.4*. <http://www.omg.org/mda/specs.htm>, 2001.
- [4] S.A. Roubtsov, E.E. Roubtsova, P. Abrahamsson. Evolutionary Product Line Modelling. *Accepted for the International Workshop on Evolution of Large-scale Industrial Software Applications (ELISA)*, Amsterdam. October, 2003.
- [5] T. Basten, W.M.P. van der Aalst. Inheritance of behaviour. *The Journal of Logic and Algebraic Programming*, 46:47-145, 2001.