

Component Specification and Composition in a UML-based tool*

E.E. Roubtsova¹, H.B.M. Jonkers², R. Kuiper¹.

¹ Eindhoven University of Technology, Department of Mathematics and Computer Science, Den Dolech 2, Postbus 513, 5600 WB Eindhoven, The Netherlands,

E.Roubtsova@tue.nl, r.kuiper@tue.nl

² Philips Research Laboratories Eindhoven, Prof. Holstlaan 4, 5656AA Eindhoven, The Netherlands,

hans.jonkers@philips.com

Abstract

We present a tool that supports both specification of a component and composition of a system from components. A component is specified as a set of roles communicating via interfaces, named an interface-suite-contract. The composition of interface-suites-contracts is defined as an inheritance of roles and a specialization of the contract by an interface-suite. A user of the tool defines and chooses in forms elements of UML diagrams corresponding to the interface-suite definition and the composition definition. The tool draws UML diagrams and generates the correspondent documentation using the information from forms. The tool is implemented as an extension of the Rational Rose UML-based tool.

1 Introduction

Using components became a real practice of programming. A component solves a standard task and the programmer who uses the component saves the time to work out an application. However, potential advantages of programming with components can be realized only if the specification of components is based on the standard definition. Otherwise, the programmer will spend even more time studying components

and understanding possible connections of components in the program [2, 7].

The Unified modeling language is the standard for system design [1]. There are several approaches to use the UML for component specification [2, 3, 6]. In this paper we follow one of the UML oriented approaches, called ISpec [3]. ISpec defines a component specification as an interaction pattern, named an interface-suite. The realization of the component can combine several roles from the pattern. *An interface-suite identifies a finite set of roles communicating via interfaces provided by these roles. An interface-suite is closed in sense that the environment is specified inside of it.* An interface-suite is represented by several UML class diagrams with the sets of the corresponding templates, specifying elements of those diagrams. Existing UML-based tools are not suitable to support the ISpec specification.

- First, the terminology of the UML have been designed for object-oriented programming and does not correspond to the interface-suite terminology. For example, an interface in the UML is an independent class, however, an interface in ISpec is a named set of operations provided by a role.
- Second, the ISpec uses Diagrams as illustrations of the specification given by specification templates.

*Supported by PROGRESS grant EES5141

- Third, the ISpec restrict composition of interface-suites by inheritance of roles.
- Fourth, the documentation structure in the ISpec is based on the definitions of an interface-suite and the composition.

In this work we describe the UML-based tool that supports both the ISpec specification of a component at the interface-role view and the composition of systems from components. The structure of the paper is the following. Section 2 defines the case study that we use to illustrate the idea of the tool. Section 3 defines an interface-suite at the interface-role view, presents the specification workflow for one interface-suite and the tool support for this workflow. Section 4 defines the composition via specialization of interface-suites at the interface-role view. The section presents also the composition workflow and the corresponding tool support. In Section 4 we describe future work.

2 Case Study

To illustrate our tool we show a specification of a hitting system. The system consists of four roles a *Controller*, a *Reflector*, a *Sensor* and a *Clock*. When the *Sensor* shows the temperature less than T , the *Sensor* calls the interface *ISwitch* of the *Controller*. The *Controller* turns the *Reflector* on via interface *ITurn*. The *Reflector* should work over τ minutes. To control this time interval the *Controller* sets the *Clock*. When the time is over, the *Controller* turns the *Reflector* off.

We construct the hitting system using component *Timer*. This component has two roles an *EggUser* and an *EggTimer*. The *EggUser* can reset the *EggTimer* to a constant. When the given time constant is reached the *EggTimer* calls interface *IReady* provided by the *EggUser* (Fig. 1).

So, first, we show how the component *Timer* is specified in our tool then we specialize this component by new graph that realize roles *Controller*, *Reflector*, *Sensor* and *Clock* such that role *Clock* inherits role *EggTimer* and *Controller* inherits role *EggUser* from *Timer* specification (Fig. 1).

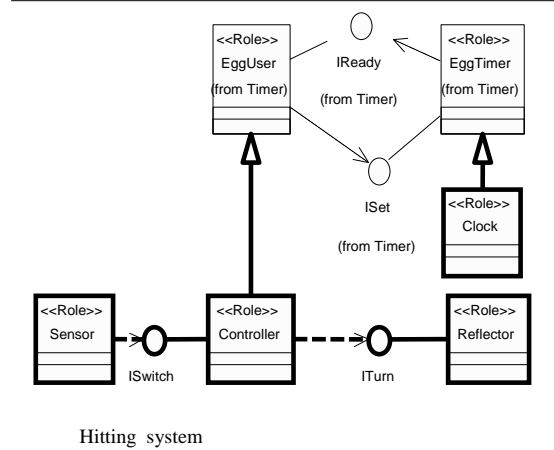
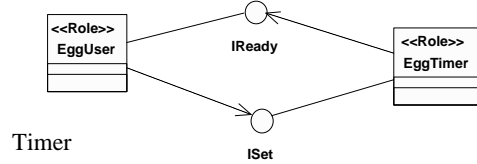


Figure 1: Interface-role diagram for a hitting system

3 Specification of an Interface-suite

At the interface-role view, an interface suite is a graph (R, I, PI, RI) with two kinds of nodes:

- R is a finite set of roles depicted by boxes; $R \neq \emptyset$.
- I is a finite set of interfaces depicted by circles, I can be empty;

and two kinds of relations

- $PI(R, I)$ defines interfaces provided by roles:

$$PI(R, I) = \{(r, i) | r \in R, i \in I\}.$$

- $RI(R, PI(R, I))$ defines interfaces required by roles.

$$RI(R, PI(R, I)) = \{(r, (r, i)) | r \in R, i \in I, (r, i) \in PI(R, I)\}.$$

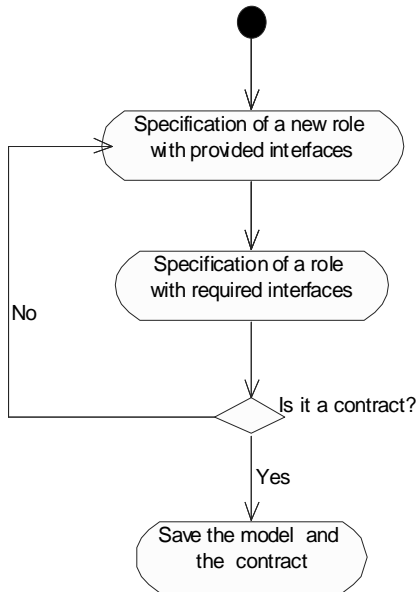


Figure 2: Specification of an interface-suite.

The relation is drawn by a dashed line with an arrow connecting a role and an interface.

Changing any of the tuple elements produces a new graph. We make a distinction between an *interface-suite* of type (R, I, PI, RI) that can be changed and an *interface-suite-contract* that can not be changed. The objective of the specification process for a component is an interface-suite-contract that can be used in composition.

The workflow in specification of an interface-suite-contract is shown on Fig. 2. A box with round corners represents an activity in the workflow. Each activity is supported by a specification form in our tool. Using information from this form some elements of interface-role diagram and the correspondent documentation in HTML format are generated by the tool.

Let us describe the activities in more detail.

- *Specification of a new role with provided interfaces.* A role of the interface-suite is specified by a Rational Rose class

$$Class = (Name, Stereotype, Documentation, Relation),$$

- the *Name* is the role name;
- the *Stereotype* is "Role";
- The *Documentation* represents the informal description, the list of responsibilities of the role as it is usual in ISpec [4, 3];
- The *Relations* set includes the provide relation *PI*) that is represented by the *class.realize* collection and the require relation *RI* that is represented by the *class.depends* list in Rose.

The independent specification of interfaces is forbidden by the tool. Interfaces can be specified via Role specification. We generate a list of all interfaces provided by roles of an interface-suite and use this list to specify the interfaces that can be required.

To specify a role we open a class at the *Interface-Role Diagram* and choose the *Open Specification* Rose context menu item instead of the *Open Standard Specification* item. We have defined another reaction to the Rational Rose Event *On-PropertySpecOpen* using Rose Extensibility interface [5]. Our Role Specification form contains tabs for *Informal specification*, *Interfaces Provided* by the Role, *Interfaces Required* by the Role.

Informal Specification tab covers the *name*, the *stereotype*, and the *documentation* of a role.

Interfaces Provided tab allows to define a new interface, provided by the current role. All interfaces provided by the role will be drawn automatically at the interface role diagram by lollipops connected with the role by solid lines. For example, the interface *IReady* is provided by the role *EggUser* (Fig. 1).

- *Specification of a role with required interfaces.*

On the *Interfaces Required* tab of the role specification the set of interfaces required by the role can be chosen from the set of the interfaces provided by roles using drag and drop. For example, interface *IReady* has been chosen by role *EggTimer* from the set of provided interfaces *ISet* and *IReady*. After the choice the arrow between role *EggTimer* and interface *IReady* appears at the diagram (Fig. 1).

Our Rational Rose ADD-IN provides special tab "Save as HTML" for creating the documentation on all specification forms in HTML-format. By clicking this tab we open the *Print Form* which allows to build the documentation as a puzzle from specification elements: roles, interfaces, operations, types. The HTML documentation has obvious advantages for internet delivery of a component that has been specified as an interface-suite. The interface role diagram is saved as a file in EMF-format in the same directory and inserted into the HTML-file of documentation.

- *Is it a contract?* If all roles and interfaces have been defined, our tool allows to save any interface-suite in two files. A Rose model file is used for changeable interface-suites. A Rose category file with extension *sui* is chosen when we make an interface-suite-contract. It is possible to save both files for every interface suite.

4 Composition of Components as Specialization of Roles

The idea of the interface-suite composition is to save the specification of composed interface-suite-contracts and the structure of the composition in the result of the composition. Such an approach helps to read and understand the composed specification. The approach allows also to correct mistakes in parents independently from children and distribute correct parent-contracts through all children-contracts.

The composition is realized via specialization of

roles. Informally, any role-child in the interface-suite always inherits all provided interfaces of its parent-roles. The inheritance of required interfaces is different. Role x_* which specializes role x from an old contract is an element a new contract. It is possible that in this new contract role x_* requires other interfaces. Therefore, the role does not automatically inherit the required interfaces from role x . So, there is the option to define new required interfaces. However, if role x_* requires the same interfaces as role x requires from role r , then some role r_* should exist in the new contract to provide those interfaces. This role r_* can be supplied through inheritance from the role r of the old contract.

A simple case of composition is where two contracts are combined to a new one and the two subsystems and no specialization occurs. Intuitively, this means that no communication between roles of those contracts occurs. This we name parallel composition.

Let us give the formal definition of the composition and specialization.

Definition (*composition of interface-suite contracts, specialization of interface suite contracts by interface-suites*)

1. Let two interface-suite-contracts C_1, C_2 be given:

$$C_1 = (R_1, I_1, PI_1(R_1, I_1), RI_1(R_1, PI_1))$$

$$C_2 = (R_2, I_2, PI_2(R_2, I_2), RI_2(R_2, PI_2)),$$

$$R_1 \cap R_2 = \emptyset, I_1 \cap I_2 = \emptyset.$$

Parallel composition of given interface-suite-contracts is an interface-suite-contract:

$$C = C_1 \parallel C_2, \quad C = (R, I, PI, RI) :$$

- $R = R_1 \cup R_2,$
- $I = I_1 \cup I_2,$
- $PI(R, I) = PI_1(R_1, I_1) \cup PI_2(R_2, I_2),$
- $RI(R, I) = RI_1(R_1, PI_1) \cup RI_2(R_2, PI_2).$

2. Let an interface-suite-contract C and an interface-suite G be given:

$$C = (R_c, I_c, PI_c(R_c, I_c), RI_c(R_c, PI_c))$$

$$G = (R_g, I_g, PI_g(R_g, I_g), RI(R_g, PI_g))$$

$$R_c \cap R_g = \emptyset, I_c \cap I_g = \emptyset.$$

Specialization of interface-suite-contract
 C_1 by interface-suite G is an interface-suite-contract:

$$S = G \triangleright C, S = (R_s, I_s, PI_s, RI_s), \text{ where}$$

- $R_s = R_c \cup R_g,$
- $I_s = I_c \cup I_g,$
- $PI_s(R_s, I_s) =$
 $= PI_c(R_c, I_c) \cup PI_g(R_g, I_g) \cup PI_*(R_*, I_*),$
 where

$$R_* \subseteq R_g, I_* \subseteq I_c,$$

$$PI_*(R_*, I_*) = \{(r_*, i) \mid r_* \in R_*, i \in I_c, \\ \exists r \in R_c, \text{ such that } r_* \triangleright r, \text{ and } (r, i) \in PI_c(R_c, I_c)\}.$$

- $RI_s(R_s, I_s) = RI_c(R_c, PI_c(R_c, I_c)) \cup$
 $\cup RI_g(R_g, PI_g(R_g, I_g)) \cup RI_*(R_*, PI_*(R_*, I_*)).$

$$RI_*(R_*, PI_*(R_*, I_*)) =$$

$$= \{(x_*, (r_*, i)) \mid r_*, x_* \in R_*, i \in I_c, \\ \exists r \in R_c, \text{ such that } r_* \triangleright r, \exists x \in R_c, \\ \text{such that } x_* \triangleright x, \\ \text{such that } ((r, i) \in PI_c(R_c, I_c) \\ \text{and } (x, (r, i) \in RI_c(R_c, PI_c))\}.$$

3. There are no other interface suite contracts.

This definition guarantees that a role can not specialize itself and parent-roles can not specialize roles-children. Both specialized roles and completely new roles belong to the changeable part, an interface suite.

The composition rules define the workflow shown in

the Fig.3. Let us explain how the activities of this workflow are supported by our tool.

- *Choice of parent interface-suite contracts.* There is a form that allows to import parent interface-suite contracts (sui-fails). The tool automatically makes those files sub-packages of the composed interface-suite and shows on the package

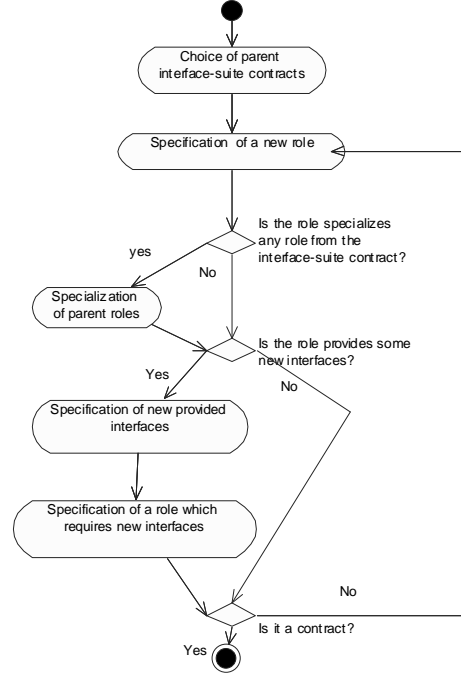


Figure 3: Composition of interface-suites

diagram. The files are used by the tool also to be represented at the interface role diagram of the composed interface suite. All roles are marked by the name of their component. For example, on Fig. 1 we can see that roles *EggTimer* and *EggUser* come from interface-suite-contract *Timer*.

- *Specification of a new role.* At the interface-role diagram when diagrams of parents have been drawn and can not be changed we can define a new role. We can put this role on the diagram and open our specification firm.
- *Specialization of a parent role.* The tab *Specialization* on our specification form for a role allows to chose parent roles from parent interfaces-suites. The specialization relation is drawn by tool at the interface-role diagram. Interfaces of parent roles inherited by the new role in the internal model. For example, role *Con-*

troller specializes role *EggUser* and inherits interface *IReady*.

- Tab *Interfaces Provided* on our specification form allows to specify new interfaces of new roles. The defined interfaces are drawn at the interface-role diagram by the tool with the provided relation between interface and its role. So, new interface *ISwitch* is defined for role *Controller*.
- *Specification of a role that requires new interfaces.* Tab *Interfaces Required* is used for a choice of required interfaces from provided. In our case, role *Sensor* has been defined that requires interface *ISwitch* of role *Controller*.
- *Is it a contract?* The designer answers this question using the form of the tool. If we need more roles to be defined, we use the cycle of the workflow (Fig.3). The alternative workflows marked by label *No* (Fig.3) allow to specify roles without specialization or without new interfaces. For example, role *Reflector* does not specialize any parent role. When all necessary roles have been defined the tool allows to save the interface-suite as a contract (sui-file) to be reused as a parent for the next interface-suite specification.

5 Conclusion

Specification of component systems is a new information technology. The success of this technology depends on development of relevant workflows with the corresponding tool support. In this paper, we have presented two of these workflows and our tool for them. We continue the development of the tool. In the new version, we are going to support another workflow: the specification of behaviour and the inheritance of this behavioural specification in composition of interface-suites. However, the current tool is already useful because

- it gives the designers of different components one graphical view on components and their place in the system;

- it specifies the set of actions for the behavioural specification in terms of interfaces provided by roles. The workflow analyses shows that this set of actions simplifies the specification of processes on UML behavioural diagrams. This set of actions allows also to define inheritance of behaviour in composition of interface-suites-contracts, that we consider as the subject for future work.

References

- [1] Booch G., Rumbaugh J., Jacobson I. *The Unified Modeling Language User Guide*. Addison-Wesley, Amsterdam, 1999.
- [2] Cheesman J., Daniels J. *UML Components. A Simple Process for Specifying Component-Based Software*. Addison-Wesley, 2001.
- [3] Jonkers H.B.M. ISpec: Towards Practical and Sound Interface Specifications. *Integrated Formal Methods*, LNCS 1945:116–135, 2000.
- [4] Jonkers H.B.M. Interface-Centric Architecture Descriptions. *In: Working IEEE/IFIP Conference on Software Architecture*, WICSA 2001:113–124, 2001.
- [5] Rational Rose 2000. *Rose Extensibility Reference 2000*.
- [6] Riehle D. *Framework Design: A Role Modeling Approach. Ph.D. Thesis, No. 13509*. Zurich, Switzerland, ETH Zurich, 2000.
- [7] Szyperski C. *Component Software Beyond Object-Oriented Programming*. ADDISON-WESLEY, New-York, 1998.