Int. J. Appl. Math. Comput. Sci., 2006, Vol. 16, No. 1, 155–164

REFINEMENT OF UML COLLABORATIONS

BOGUMIŁA HNATKOWSKA, ZBIGNIEW HUZAR
LECH TUZINKIEWICZ

Institute of Applied Informatics, Wrocław University of Technology
ul. Wybrzeże Wyspiańskiego 27, 50–370 Wrocław, Poland
e-mail: {bogumila.hnatkowska,zbigniew.huzar,
lech.tuzinkiewicz}@pwr.wroc.pl

The paper concerns the concept of refinement as a commonly used design practice in the software development process. The refinement relationship is formulated and formally expressed for UML collaborations. Collaborations are suitable for presenting the proposed approach as they represent both static and dynamic aspects of a modeled system or its part, for example, a use case. Our approach to refinement is based on the rule of preserving the observable behavior of a modeled system. The introduced notion of sub-collaborations allows us to refine collaboration diagrams in a systematic way.

Keywords: UML, collaborations, refinement

1. Introduction

Refinement is a classic technique used in software design that may also be applied to the elaboration of successive models in the software development process based on the object-oriented paradigm. Nowadays, the UML is a modeling language widely used in object-oriented software development (Fowler and Scott, 2000). The refinement relationship is introduced in the UML as a standard stereotype "refine" of the abstraction relationship. The glossary of terms in both UML 1.x and UML 2.0 describes refinement as (OMG, 2003):

> *A relationship between model elements at different semantic levels, such as analysis and design. The mapping specifies the relationship between the two elements or sets of elements. The mapping may or may not be computable, and it may be unidirectional or bidirectional. Refinement can be used to model transformations from analysis to design and other such changes.*

In the book (Rumbaugh *et al.*, 2004), we have an additional explanation:

> *A relationship that represents a fuller specification of something that has already been specified at a certain level of detail or at a different semantic level. A refinement is a historical or computable connection between two elements with a mapping (not necessarily complete) between them. Often, the two elements are in different models. For example, a design class may be a refinement of an analysis class; it has the same logical attributes, but their classes may come from a specific class library. An element can refine an element in the same model, however. For example, an optimized version of a class is a refinement of the simple but inefficient version of the class. The refinement relationship may contain a description of the mapping, which may be written in a formal language (such as OCL or a programming or logic language). Or it may be informal text (which, obviously, precludes any automatic computation but may be useful in early stages of development). Refinement may be used to model stepwise development, optimization, transformation, and framework elaboration. Refinement is a kind of abstraction dependency. It relates a client (the element that is more developed) to a supplier (the element that is the base for the refinement).*

The explanation points that refinement is a kind of abstraction dependency but, unfortunately, the abstraction relationship is also informally defined (OMG, 2003):

> *An abstraction is a dependency relationship that relates two elements or sets of elements that represent the same concept at different levels of abstraction or from different viewpoints.*

The definitions allow many interpretations but do not give methodological hints or suggestions how to refine one model into another or how to check that one model is a refinement of another. The paper is a contribution towards the formalization of the notion of the refinement relationship.

Refinement is a notion extensively applied to the software development process. Manassis (2004) de-

fines software engineering as "refinement of knowledge through successive abstraction levels of representation", and "traceability of each and every item of information between abstraction levels". The refinement relation is one of the forms of representing changes within the system under development, and it involves progressive focusing on the details of the problem being solved. The developing of a system using refinement starts with a set of fundamental concepts, and advances towards a more concrete implementation in a stepwise fashion. The subsequent models are decomposed into ones that are more concrete.

Modern methodologies, e.g., MDA (Hubert, 2001), USDP (Jacobson *et al.*, 1999), RUP (Krutchen, 1999) present the software development as a process of model refinement. Therefore, there is a need to formally define this notion. Our aim was to define the refinement based on the UML semantics.

As the UML offers a great number of modeling elements, we had to restrict our consideration to a possible small set of modeling elements that can be used to build reasonable models. Therefore, we have decided to choose collaborations as a subject of our consideration as they enable us to represent both static and dynamic aspects of a modeled system.

The collaboration diagrams presented in this paper purposefully exclude n-ary connectors (for $n > 2$) as well as interfaces, since their absence would not affect the validity of the proposed definition of refinement, while causing an unnecessary expansion of the formal definition (irrelevant from the point of view of the issue at hand).

Collaborations are used to describe the realization of services represented by use cases or operations. A service is perceived as set of interactions, i.e., a partially ordered set of communications between service users and service providers. So, a refinement of a collaboration should provide a more detailed description of the interactions in the context of the collaboration structure.

The paper presents a formal approach to defining refinement, enabling its unambiguous interpretation. We have decided to use the set theory language for the formal description of models as it seems to be the most suitable in the context of the UML. In the UML, the metaclass *M*odel is a specialization of the metaclass *P*ackage, which gathers different kinds of model elements (OMG, 2003).

> *The model construct is defined as a package. It contains a (hierarchical) set of elements that together describe the physical system being modeled. A model may also contain a set of elements that represents the environment of the system, typically Actors, together with their interrelationships, such as associations and dependencies.*

Moreover, UML case tools represent a model as a set of system elements.

The refinement may be considered as:

- an enrichment of the descriptions of the existing entities (i.e., new properties) at a given semantic level, or
- developing structures and behavior through the introduction of new, interconnected entities as refinements of the existing ones (transformation between different semantic levels).

In the paper we omit the enrichment of the descriptions of the existing entities since it was a subject of other works, e.g., (Hnatkowska *et al.*, 2003; Liu *et al.*, 2004a), concentrating on the structure and behavior developing.

The notion of the semantic level was defined in (Hnatkowska *et. al.*, 2004c) and is expressed in terms of classifiers and relationships between them.

The paper is organized as follows: Section 2 reminds us very briefly what UML collaborations are. Section 3 gives the definition of collaboration diagrams, while Section 4 provides the definition of communication diagrams. Sections 5 and 6 formally define the refinement of collaboration and communication diagrams, respectively. A simple example illustrating our approach to refinement is given in Section 7. Finally, in Section 8, we present our conclusions.

2. UML Collaborations

A collaboration is a modeling element that is used to explain how a system works. Generally, it describes a collection of assembled objects that interact to implement some behavior within a context. Usually, in a software development process, a collaboration is used to describe how a use case or operation are realized.

The collaboration is a specific kind of classifier (more precisely, a structured classifier), which is not directly instantiable as its elements are a view (or a projection) of instances of a context – either already defined classifiers or just anticipated classifiers. Apart from collaboration instances, we talk about collaboration occurrences that are a particular use of a collaboration to explain the temporal contextual relationships between the parts of a classifier or the properties of an operation. The collaboration is a selective view of a situation (a snapshot) of a set of cooperating classifiers.

The description of a collaboration in UML 2.0 is expressed in terms of roles and connectors participating in performing a specific task. The notion of the connector replaces the notion of an association role in UML 1.x.

A role is a named set of features defined over a collection of entities participating in a particular context. In practice, the role is defined as a view of a class through

its selected operation. This means that such a role is a subset of the properties of the class which is related to the selected operation. The roles are filled by objects at run time.

A connector specifies a link that permits communication between two or more instances. Links specified by connectors represent communications paths among the parts of a collaboration. The connector may be a projection of an already defined association, or it may represent the possibility of objects being able to communicate because the objects obtain information about their identities in an implicit way.

Both roles and connectors are classifiers. Let us recall that a classifier specifies a universe of its instances. Further, we will use the notion *Univ*(C) meaning the universe of instances of the classifier C. The definition of the universe is a counterpart of an extensionally defined set. An instance c of the classifier C will be denoted by $c : C$.

The collaboration itself is only a description of a temporal structure of a behavior provider. Its structure may be a base for a definition of behavior – interactions that occur during a use case or an operation realization.

An interaction is a specification of how messages are sent between the elements of a collaboration to perform a specific task. The message is a one-way communication between two objects, a flow of control with conveying values from a sender to a receiver. The message can be a signal (an explicit, named, asynchronous inter-object communication) or a call (a synchronous or an asynchronous invocation of an operation with a mechanism to return later control to the sender of the synchronous call).

An instance of an interaction corresponds to an instance of its context, with objects bound to roles exchanging message instances across links bound to connectors. So, the collaboration may have attached a set of behavior that apply to a set of objects bound to a single instance of the context.

Collaborations are represented by collaboration diagrams while interactions may be represented by communication diagrams, sequence diagrams or interaction overview diagrams. Sequence diagrams and communication diagrams express similar information, but show it in different ways. In the sequel, we consider a collaboration represented by a collaboration diagram and a set of attached communication diagrams.

The communication diagram shows interactions organized around the roles of a collaboration. It is a collaboration instance together with messages that forms an interaction. The sequencing of messages is given through a sequence numbering scheme.

The UML formalizes the syntax for collaborations and interactions. Their semantics is defined informally in a natural language. A collaboration diagram *CollD* with a related commucation diagram *CommD* represent a structural and behavioral aspect of a model of a modeled system. By $\mathit{Coll} = \langle \mathit{CollD}, \mathit{CommD} \rangle$ we will denote the integration of both diagrams, called the collaboration model. For the sake of simplicity, we assume that only one interaction represented by a communication diagram is attached to the collaboration diagram.

3. Collaboration Diagrams

We assume that roles are a projection of classes, and we denote by $\boldsymbol{C}$ a finite set of role names. To simplify the presentation, we assume that operations owned by roles have fully specified signatures, and they have public visibility. Let $\boldsymbol{O}$ be a set of operations of roles. Each role $C \in \boldsymbol{C}$ is given a set of operations $\boldsymbol{LC}(C) \subseteq \boldsymbol{O}$ that may be called by other roles. If $C \in \boldsymbol{C}$, then $op : C$ means that *op* is an operation of the role C.

Operations may be invoked in synchronous or asynchronous modes while signals only asynchronously. Therefore, we may omit signals without any loss of generality.

We denote by $\boldsymbol{A}$ a finite set of connectors. Each connector $A \in \boldsymbol{A}$ connects a set of roles $\boldsymbol{LA}(A) \subseteq \boldsymbol{C}$. If $A \in \boldsymbol{A}$, then $C \in \boldsymbol{LA}(A)$ means: C is the role at an end of the connector A. Further, we assume that binary connectors appear only as a communication in the UML, except signal broadcasting, occurs between two objects.

The collaboration diagram *CollD*, representing a structural aspect of a collaboration model, is defined as the quintuple

$$\mathit{CollD} = \langle \boldsymbol{C}, \boldsymbol{O}, \boldsymbol{LC}, \boldsymbol{A}, \boldsymbol{LA} \rangle.$$

For a collaboration diagram *CollD* we define its collaboration sub-diagram $\mathit{CollD}' = \langle \boldsymbol{C}', \boldsymbol{O}', \boldsymbol{LC}', \boldsymbol{A}', \boldsymbol{LA}' \rangle$, which is denoted by $\mathit{CollD}' \subseteq \mathit{CollD}$, as the collaboration diagram for which the following holds:

(i) $\boldsymbol{C}' \subseteq \boldsymbol{C}$,

(ii) $\boldsymbol{O}' \subseteq \boldsymbol{O}$,

(iii) $\boldsymbol{LC}' = \boldsymbol{LC}|_{\boldsymbol{C}'}$,

(iv) $\boldsymbol{A}' \subseteq \boldsymbol{A}$,

(v) $\boldsymbol{LA}' = \boldsymbol{LA}|_{\boldsymbol{A}'}$,

where $\boldsymbol{LC}|_{\boldsymbol{C}'}$ means a restriction of the function $\boldsymbol{LC}$ to its sub-domain $\boldsymbol{C}' \subseteq \boldsymbol{C}$. The set defined as follows:

$$\mathit{rel}A(\mathit{CollD}') = \{A \in \boldsymbol{A} \backslash \boldsymbol{A}' \mid \boldsymbol{LA}(A) \cap \boldsymbol{C}' \neq \emptyset\}$$

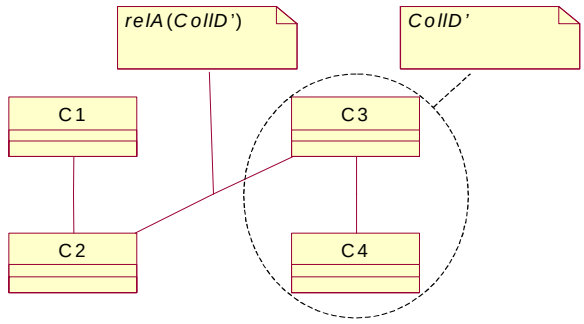

Fig. 1. Set of connectors relevant to the collaboration sub-diagram $CollD'$ in the collaboration $CollD$.

will be called the set of connectors relevant to the collaboration sub-diagram $CollD'$ in the collaboration diagram $CollD$, see Fig. 1.

Let $C \in \boldsymbol{C}$ be an arbitrary selected role in the collaboration diagram $CollD$. Now, we define a minimal collaboration sub-diagram

$$CollD_C = \langle \boldsymbol{C}_C, \boldsymbol{O}_C, \boldsymbol{LC}_C, \boldsymbol{A}_C, \boldsymbol{LA}_C \rangle$$

of the diagram $CollD = \langle \boldsymbol{C}, \boldsymbol{O}, \boldsymbol{LC}, \boldsymbol{A}, \boldsymbol{LA} \rangle$ with respect to the role C.

$CollD_C$ is the collaboration sub-diagram satisfying the following conditions:

(i) $\boldsymbol{C}_C = \{C' \in \boldsymbol{C} \mid \exists A \in \boldsymbol{A} \bullet \boldsymbol{LA}(A) = \{C, C'\}\} \cup \{C\}$,

(ii) $\boldsymbol{O}_C = \bigcup_{C' \in \boldsymbol{C}_C} \boldsymbol{LC}(C')$,

(iii) $\boldsymbol{LC}_C = \boldsymbol{LC}|_{\boldsymbol{O}_C}$,

(iv) $\boldsymbol{A}_C = A' \in \boldsymbol{A} \mid C \in \boldsymbol{LA}(A')$,

(v) $\boldsymbol{LA}_C = \boldsymbol{LA}|_{\boldsymbol{A}_C}$.

Now, we define an instance of a collaboration diagram $CollD$ as the pair

$$Inst\,CollD = \langle Ins\,C,\ InsA \rangle,$$

where $Ins\,C$ is a set of instances of the roles $\boldsymbol{C}$, and $InsA$ is a set of instances of the connectors $\boldsymbol{A}$, if the following conditions are satisfied:

(i) $Ins\,C \subseteq \bigcup\limits_{C \in C} Univ(C)$,

(ii) $InsA \subseteq \bigcup\limits_{A \in A} Univ(A)$,

(iii) if $a \in InsA$ then $LA(a) \subseteq Ins\,C$,

where $LA(a)$ means a set of role instances at the ends of the link a. Let $o : C$ mean that o is an instance of the role C, and $a : A$ mean that a is a link specified by the connector A. The function LA corresponds to the function $\boldsymbol{LA}$ as follows: if $a : A$ and $c : C \in LA(a)$, then $C \in \boldsymbol{LA}(A)$. In a similar way, the function LC corresponds to the function $\boldsymbol{LC}$ as follows: if $c : C$, then $LC(c) = \boldsymbol{LC}(C)$.

In the same way as for a collaboration diagram, we introduce an instance collaboration sub-diagram for an instance diagram.

Let $InstCollD = \langle Ins\,C, InsA \rangle$ be the instance collaboration diagram. The pair $InstCollD' = \langle Ins\,C', InsA' \rangle$, where $Ins\,C' \subseteq Ins\,C$ and $InsA' \subseteq InsA$, is called an instance collaboration sub-diagram, which is denoted by $InstCollD' \subseteq InstCollD$, if it is an instance collaboration diagram. So, the set of links relevant to the instance collaboration sub-diagram is defined as follows:

$$relA(InsCollD') = \{a \in InsA \backslash InsA' \mid LA(a) \cap Ins\,C' \neq \emptyset\}.$$

A minimal instance collaboration sub-diagram $InstCollD_c = \langle Ins\,C_C, InsA_c \rangle$ with respect to the instance c is defined as follows:

(i) $Ins\,C_c = \{c' \in Ins\,C \mid \exists a \in InsA \bullet LA(a) = \{c, c'\}\} \cup \{c\}$,

(ii) $InsA_c = \{a' \in Ins_A \mid c \in LA(a')\}$.

4. Communication Diagrams

A communication diagram is an instance of the collaboration diagram with links labeled by a set of messages which are partially ordered. We define a communication diagram $CommD$ as the triple

$$CommD = \langle InsCollD,\ Comm,\ succ \rangle,$$

where $InstCollD = \langle Ins\,C, InsA \rangle$, $Comm$ is a set of communications, and $succ \subseteq Comm^2$ is a successor relation among communications. Instances of roles communicate to each other transmitting messages, see Fig. 2.

A message is the transmission of a signal from one instance to one or more carried out instances, or it is a call of an operation on one instance by another instance. In what follows, we assume that only operation calls are considered.

A communication $com \in Comm$ is the triple

$$com = \langle s : S,\ r : R,\ op : LC(R) \rangle,$$

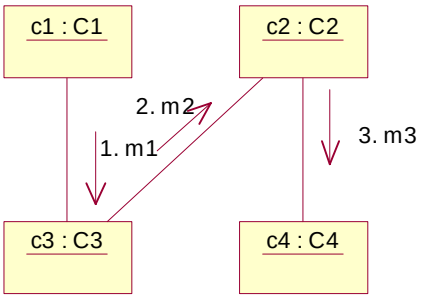

Fig. 2. Communication diagram.

where $s : S,\ S \in \boldsymbol{C}$, is an instance sending the message *op* being a call of an operation of a receiving instance $r : R$. By definition,

$$com.send = s : S,\ com.rec = r : R,$$

and

$$com.call = op : LC(R).$$

The relation $succ \subseteq Comm^2$ is a causality relation among the set of communications *Comm*. It is defined as follows: if $com_1 succ\, com_2$, then the message $com_1.call$ is a direct predecessor of the message $com_2.call$ (or the message $com_2.call$ is a direct successor of the message $com_1.call$). The interpretation of the relation means that a given message is sent only if all its direct predecessor messages have been sent.

The relation *succ* satisfies the following feasibility condition:

$$\text{if}\quad com_1\ succ\ com_2\quad \text{then}\ com_1.rec = com_2.send,$$

i.e., the sender of the message $com_2.send$ is the receiver of the message $com_1.call$.

We denote by $succ^*$ the transitive closure of the relation *succ*. It is easy to check that $succ^*$ is a relation of partial ordering. Therefore, we define the set of minimal elements for the relation, denoted by $succ^*_{\min}$, as follows:

$$succ^*_{\min} = \{com \in Comm \mid \neg\exists com' \in Comm \bullet com'\, succ^*\, com\}.$$

Let us define the function $LM : InsA \to 2^M$ (where M is a set of messages), which assigns a set of communicates to a given link:

$$LM(l) = \{com \mid com.send \in LA(l) \vee com.rec \in LA(l)\}.$$

For a communication diagram *CommD* we define its communication sub-diagram *CommD′*, denoted by $CommD' \subseteq CommD$, as a communication diagram for which the following holds:

(i) $Ins\,C' \subseteq Ins\,C$,

(ii) $InsA' \subseteq InsA$,

(iii) $Comm' \subseteq Comm$,

(iv) $succ' = succ|_{Comm' \times Comm'}$.

A communication sub-diagram of the communication diagram *CommD* is called a minimal communication sub-diagram with respect to an instance c of the role C, denoted by $CommD_c = \langle Ins\,C_c, InsA_c, Comm_c, succ_c\rangle$, if the following holds:

(i) $Ins\,C_c = \{c' \in Ins_C \mid \exists l \in InsA \bullet LA(l) = \{c, c'\}\} \cup \{c\}$,

(ii) $InsA_c = \{a \in InsA \mid c \in LA(a)\}$,

(iii) $Comm_c = \bigcup_{a \in InsA_c} LM(a)$,

(iv) $succ_c = succ|_{Comm_c \times Comm_c}$.

We may say that a communication sub-diagram of a given communication diagram is generated by a selected subset of instances. In this sense a minimal communication sub-diagram with respect to an instance c is a communication sub-diagram generated by this instance.

5. Refinement of Collaboration Diagrams

A refinement of a collaboration diagram $CollD_1$ into a new one $CollD_2$ is concerned with a transition from an abstract to a more detailed semantic level. A more detailed semantic level reveals new roles and connectors. The idea behind our approach to refinement results from the assumption that a given level of abstraction is expressed mainly in terms of selected classifiers (Hnatkowska *et al.*, 2004c). In the presented approach, similarly as is done in (Harel and Politi, 1998), we have chosen roles as the basic classifiers. Connectors between roles are taken as a consequence of role selection. In this sense connectors, as compared to roles, play a secondary role. So, the refinement of a collaboration means a refinement of the set of its roles.

We follow a systematic approach based on the selection of a collaboration sub-diagram $CollD' \subseteq CollD_1$, and replacing it by another more detailed collaboration sub-diagram $CollD''$. As a result of this replacement, we obtain a new collaboration diagram $CollD_2$. The sub-diagrams $CollD'$ and $CollD''$ should be consistent. Below, we define consistency conditions for the replacement of the minimal collaboration sub-diagram with respect to a selected role C in the collaboration diagram $CollD_1$.

Let

$$CollD_C = \langle \boldsymbol{C}_C, \boldsymbol{O}_C, \boldsymbol{LC}_C, \boldsymbol{A}_C, \boldsymbol{LA}_C\rangle$$

be the replaced collaboration sub-diagram, and

$$CollD'_C = \langle \boldsymbol{C}'_C, \boldsymbol{O}'_C, \boldsymbol{LC}'_C, \boldsymbol{A}'_C, \boldsymbol{LA}'_C \rangle$$

be the replacing collaboration sub-diagram.

The following conditions must hold:

(a) $\boldsymbol{C}_C \backslash C \subseteq \boldsymbol{C}'_C$,

(b) $\bigcup_{C' \in C_C} \boldsymbol{LC}_C(C') \subseteq \bigcup_{C' \in C'_C} \boldsymbol{LC}'_C(C')$,

(c) $\forall \boldsymbol{C}' \in \boldsymbol{C}'_C \backslash C_C \bullet \exists A \in A'_C \bullet LA'_C(A) = \{C', C''\} \wedge C' \neq C''$,

(d) $\forall A \in \boldsymbol{A}_C \bullet \exists A' \in \boldsymbol{A}'_C \bullet \exists C' \in C'_C \backslash \boldsymbol{C}_C \bullet \exists C'' \in \boldsymbol{C}_C \backslash \{C\} \bullet \boldsymbol{LA}'_C(A) = \{C', C''\}$.

The conditions define the structural consistency between two collaboration diagrams. The first condition means that the selected role C is replaced by a set of new roles, and the remaining roles of $CollD_C$, i.e., the set $\boldsymbol{C}_C \backslash C$ must be preserved. For example, in Fig. 3, the role $C2$ is replaced by the set of roles $C5, C6$, and the remaining roles are preserved. The newly introduced roles should perform all operations performed by the selected role C (condition (b)). The newly introduced roles may not stand alone, they should be directly or indirectly connected to the preserved roles (condition (c)) – in Fig. 3 role $C6$ is connected to the roles $C1$ and $C3$. Each connector in $CollD_C$ is replaced by at least one connector in $CollD'_C$ which has a connection to the preserved roles (condition (d)). For example, the association $a1$ is mapped to $a1'$ and the association $a2$ is mapped to $a2'$.

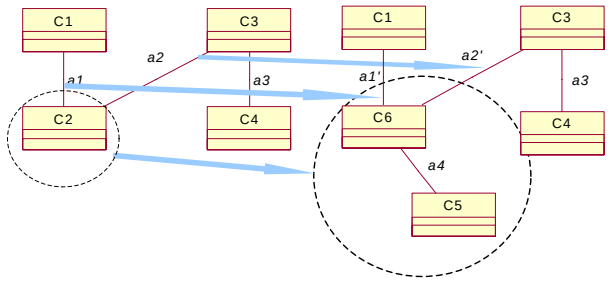

Fig. 3. Structural refinement of collaboration diagrams.

So, finally, the newly developed collaboration diagram $CollD_2 = \langle \boldsymbol{C}_2, \boldsymbol{O}_2, \boldsymbol{LC}_2, \boldsymbol{A}_2, \boldsymbol{LA}_2 \rangle$ may be defined as

(i) $\boldsymbol{C}_2 = \boldsymbol{C}_1 \backslash C \cup \boldsymbol{C}_{C'}$,

(ii) $\boldsymbol{A}_2 = \boldsymbol{A}_1 \backslash \boldsymbol{A}_C \cup \boldsymbol{A}_{C'}$.

Observe that $\boldsymbol{A}_C \cap \boldsymbol{A}_{C'} = relA(CollD_{C'})$:

(iii) $\boldsymbol{O}_2 = \boldsymbol{O}_1 \cup \boldsymbol{O}_{C'}$,

(iv) $\boldsymbol{LC}_2 = \boldsymbol{LC}_1|_{C \backslash \{C\}} \cup \boldsymbol{LC}_{C'}$,

(v) $\boldsymbol{LA}_2 = \boldsymbol{LA}_1|_{relA(CollD_C)} \cup \boldsymbol{LA}_{C'}$.

Now, we define the refinement of an instance collaboration diagram. We say that an instance collaboration diagram $InsCollD_2$ is the refinement of an instance collaboration diagram $InsCollD_1$ if $InstCollD_2$ and $InstCollD_1$ are instances of collaboration diagrams $CollD_2$ and $CollD_1$, respectively, provided that $CollD_2$ is a refinement of $CollD_1$.

6. Refinement of Communication Diagrams

The refinement of a communication diagram $CommD_1 = \langle InsCollD_1, Comm_1, succ_1 \rangle$ into a communication diagram $CommD_2 = \langle InsCollD_2, Comm_2, succ_2 \rangle$ consists of two transformations:

- refinement of the instance collaboration diagram $InsCollD_1$ into the instance collaboration diagram $InsCollD_2$,
- refinement of the set of communication $Comm_1$ partially ordered by the relation $succ_1*$ into the set of communication $Comm_2$ partially ordered by the relation $succ_2^*$.

The first transformation has already been defined, the second transformation is defined as follows.

Let $InsCollD_1$ be an instance collaboration diagram refined to an instance collaboration diagram $InsCollD_2$. Further, let $InsCollD_c$ be a minimal instance collaboration sub-diagram with respect to an instance c in $InsCollD_1$, which is refined by an instance collaboration sub-diagram $InsCollD'_c$ in $InsCollD_2$.

For each $com \in Comm_c$ there is exactly one $com' \in Comm'_c$ such that:

(a) if $com.send = c$, then $com'.rec = com.rec$ and $com.call = com'.call$,

(b) if $com.rec = c$, then $com'.send = com.send$ and $com.call = com'.call$.

It is easy to note that each communication $com \in Comm_c$ considered above labels a link relevant to $InsCollD_c$.

By $map(com) \in Comm'_c$ we denote communication that corresponds in that way to $com \in Comm_c$, and

$$map(Comm_c) = \{com' \in Comm'_c \mid \exists com \in Comm_c \bullet map(com) = com'\}.$$

We impose the following requirements:

(c) if $com_1 \neq com_2$, then $map(com_1) \neq map(com_2)$,

(d) $\forall com_1, com_2 \in Comm_c \bullet com_1 \ succ_c^* \ com_2 \Rightarrow map(com_1) \ succ_c'^* \ map(com_2)$,

(e) $\forall com' \in Comm'_c \backslash map(Comm_c) \bullet \exists com \in Comm_c \bullet (com' \ succ_c'^* \ map(com) \vee map(com) \ succ_c'^* \ com')$.

The conditions (a), (b) and (c) define a one-to-one mapping between the set $Comm_c$ and a subset of $Comm'_c$. The condition (d) requires that the ordering among communications $Comm_c$ be preserved by communications being their images in the set $Comm'_c$. The condition (e) says that each communication in $Comm'_c$ which is not a mapping of a communication from $Comm_c$ is caused by a communication that corresponds to a communication from $Comm_c$, or it causes an issue of a communication that corresponds to a communication from $Comm_c$. The last condition may be more precise:

$com' \ succ_c'^* \ map(com)$
holds provided that $com.send \neq c$,

and $map(com) \ succ_c'^* \ com'$
holds provided that $com.send = c$.

Finally, as a result of the refinement of the new communication diagram,

$$CommD_2 = \langle InsCollD_2, Comm_2, succ_2 \rangle,$$

where

(i) $InsCollD_2$ is a refinement of the instance collaboration diagram $InsCollD_1$, and

(ii) $Comm_2 = Comm_1 \backslash Comm_c \cup Comm'_c$,

(iii) $succ_2 = succ_1|_{(Comm_1 \backslash Comm_c)^2} \cup succ_c'$.

7. Example

Now, we illustrate the process of refinement for the collaboration diagram presented in Fig. 4(a). There are two classes (roles) in the diagram: *Actor* and *System*. The communication diagram being an instance of the collaboration diagram is presented in Fig. 4(b). Communication consists of two messages involving *prepare*, and print operations, sending by the *Actor* to the *System*. The operation *prepare* precedes the operation *print*.

The class *System* is used to point out the minimal sub-collaboration. This class is split into two classes: *ReportController*, and *Report*, see Fig. 5(a). The new classes are associated with the class *Actor*, and they are also associated with each other. Communication between the classes is extended with a new message with the operation *create*, which is sent from the class *ReportController* to the class *Report*, see Fig. 5(b). This new message precedes the message involving the *print* operation.

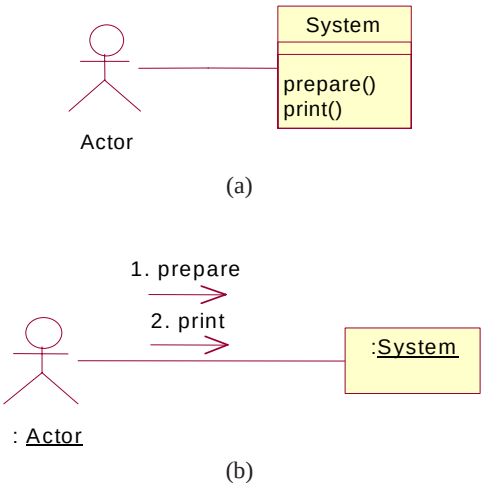

Fig. 4. Examplary collaboration diagram (a) and an examplary communication diagram (b).

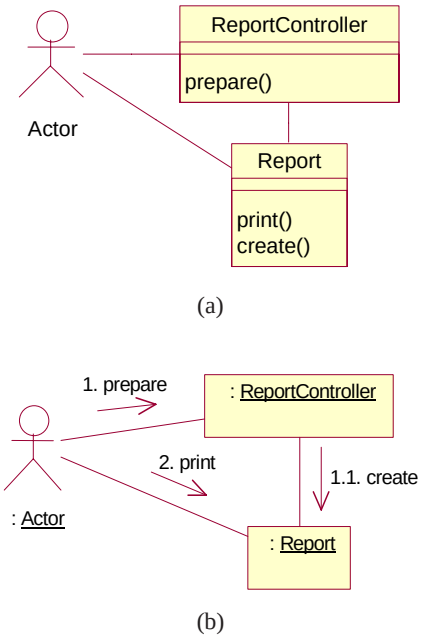

Fig. 5. Collaboration diagram after refinement – the first step (a), and a communication diagram after refinement – the first step (b).

All conditions defined in Section 5 for the structural refinement of collaboration diagrams are fulfilled:

(a) $\boldsymbol{C}_C \backslash \{\boldsymbol{C}\} \subseteq CC'$ – the class *System* is replaced by two classes: *ReportController*, and *Report*; the rest (i.e., the class *Actor*) is preserved,

(b) $\bigcup_{C' \in C_C} \boldsymbol{LC}_C(C') \subseteq \bigcup_{C' \in C'_C} \boldsymbol{LC}'_C(C')$ – the operations performed by the class *System* (i.e., *prepare*, and *print*) are now performed by the newly introduced classes,

(c) $\forall \boldsymbol{C'} \in \boldsymbol{C'_C} \backslash \boldsymbol{C_C} \bullet \exists A \in \boldsymbol{A'_C} \bullet \boldsymbol{LA'_C}(A) = \{C', C''\} \wedge C' \neq C''$ – the newly introduced classes are associated with the preserved ones, i.e., the *Report,* and *ReportController* classes are associated with the class *Actor*,

(d) $\forall A \in \boldsymbol{A_C} \bullet \exists A' \in \boldsymbol{A'_C} \bullet \exists C' \in \boldsymbol{C'_C} \backslash \boldsymbol{C_C} \bullet \exists C'' \in \boldsymbol{C_C} \backslash \{C\} \bullet \boldsymbol{LA'_C}(A) = \{C', C''\}$ – the only connector from the collaboration diagram from Fig. 4(a) is replaced by two connectors in Fig. 5(a), which has a connection to the preserved classes, i.e., the connector between the classes *Actor* and *Report,* and the connector between the classes *Actor* and *ReportController*.

In the next step, we decide to split the *ReportController* class into the *ReportController* and *Item* classes. The communication diagram is extended with a sequence of messages sent to objects of the class *Item*, see Fig. 6.

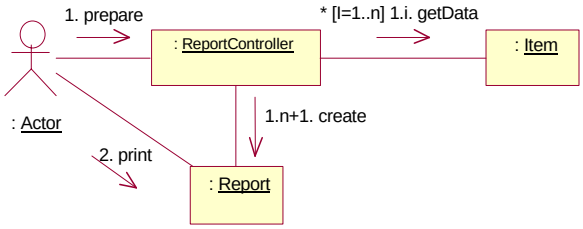

Fig. 6. Example of the communication diagram after refinement – the second step.

Let *ReportController* be the instance to construct a minimal instance collaboration sub-diagram. The communications that should be considered are:

$$com_1 = \langle :Actor, :ReportController, prepare \rangle,$$
$$com_2 = \langle :ReportController, :Report, create \rangle,$$

and $com_1 succ\ com_2$.

Now, we can check if all the conditions defined in Section 6 for behavioral refinement are fulfilled:

(a)–(b) These conditions define a one-to-one mapping between the set of communications from the original one and refined communication diagrams. For example, for the communication com_1, which is received by the *ReportController* instance, there is exactly one $com'_1 = \langle :Actor, :ReportController, prepare \rangle$ such that $com'_1.send = com_1.send$ and $com_1.call = com'_1.call$. Similarly, for the communication com_2, which is sent by the *ReportController*, instance, there exists exactly one $com'_2 = \langle :ReportController, :Report, create \rangle$ such that $com'_2.rec = com_2.rec$ and $com_2.call = com'_2.call$.

(c) Of course, $map(com_1) \neq map(com_2)$.

(d) $com'_1 succ^* com'_2$, because the operation *prepare* is performed as the first operation, and the operation *create* – after the *prepare* one (see Fig. 6).

(e) The communication diagram in Fig. 6 introduces the set of new communications com', i.e., $[i = 1, \ldots, n]$ $1.i$ *getData*. The ordering of communications is such that for each such communication there exist com_1, com_2 for which $com_1\ succ^* com'$ and $com'\ succ^* com_2$.

8. Related Works and Conclusions

The refinement relationship appears in numerous publications (Clark, 2000; Hnatkowska *et al.*, 2004b; 2004c; Liu *et al.*, 2004b; Pons *et al.*, 2000). Most authors make use of this notion to describe the software engineering process treating refinement as an element of the proposed (or described) methodologies (Souza and Wills, 1999). Yet, hardly anyone devotes much effort to further analyze this notion, preferring instead to rely on its "common sense" understanding (Katara and Mikkonen, 2002).

There are several formal approaches to the refinement definition. They depend on the semantic representation of UML models. For example, in (Lano and Bicarregui, 1999), the semantics of class diagrams is given in terms of theories in a first-order logic, and therefore a structural refinement of class diagrams is presented by means of Z-notation, while the behavioural refinement of class diagrams – by means of real-time action logic. Ordered labelled multisets based on (Pratt, 1986) are used in (Cengarle and Knapp, 2004) for the representation of UML semantics and refinement of interactions. This mathematically elegant work takes only the behavioural aspect of a model into account. Another example of behaviour refinement omitting structural issues can be found in (Clark, 2000), where the author defines refinement using lambda-calculus.

Boiten and Bujurianu (2003) define refinement as a relationship between two existing models. In this approach the refinement is seen as a kind of interconsistency relationship between models. This is in contrast to our approach, where successive models are obtained as a result of transformation/mapping based on the refinement relationship.

In the work (Pons and Kutche, 2004), there is a description of a number of refinement patterns that are implemented in the PAMPERO tool which is integrated in the Eclipse environment. Refinement transformations proposed in the paper take into consideration only one semantic level, which means that one can modify only the existing model elements. This proposal is an example of a

pragmatic and informal approach to refinement. In our paper we define refinement as a set of transformation rules. The application of the rules to a model yields a new refined model.

In (Egyed, 2002), refinement is defined to be a reverse of abstraction – a process that transforms lower-level elements into higher-level elements. The relationship is considered for class diagrams without any class properties (operations, attributes). The author defines a set of rules for models transformation/views at different abstraction levels. The rules were implemented in the UML/Analyzer tool integrated with Rational Rose.

Our approach considers refinement at two semantic levels. Similarly, different semantic levels are assumed in (Boiten and Bujorianu, 2003; Egyed, 2002; Liu *et al.*, 2004b).

This paper presents a formalization of collaboration diagrams, and then proposes transformations allowed within the process of collaboration refinement. The introduced notion of sub-collaborations allows us to systematically refine collaboration diagrams, i.e., first, to select a well-defined fragment of a collaboration (a subcollaboration) – a set of classes and a respective set of associations together with the behavior assigned to the fragment, and, next, to transform the fragment preserving its observable behavior.

Our formalization of collaboration refinement supports building up models in a stepwise fashion. However, we did not propose practical guidelines for using refinement transformations for collaboration, since they are closely bound to the methodology employed during software development.

In this paper we used the set theory language to describe UML models. The approach seems to be well suited to the software developer's needs. Graph theory can also be applied to a formal definition of different relationships between UML models, for example, we used this approach in (Hnatkowska *et al.*, 2004a). The complexity of both approaches is similar.

The formalization – the set of axioms and rules operating over finite sets – forms a base for the construction of algorithms that may check whether two collaborations remain in a refinement relationship. Such algorithms might be pluged into the existing UML-based software development tools, e.g., Rational Rose, HUGO (Knapp and Merz, 2002; Knapp *et al.*, 2002), Rhapshody, etc. A developer could define a trace relationship between two subcollaborations in the models at different semantic levels, and then the tool could check if these two models are in a refinement relationship.

Our formalization also enables us to distinguish between refinement and other kinds of dependency relationships, for example, refactoring or extension, etc. In some works, e.g., (Pons and Kutche, 2004), refinement and refactoring are not clearly discriminated. Usually, refactoring is concerned with code changing. It is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. When you refactor, you improve the design of code after it has been written (Fowler *et al.*, 1999). Refactoring involves reorganizing the structure of an existing element or solution to make it easier to understand, use and maintain without changing its observable behavior. It should be noted that refactoring was thoroughly addressed in (Sunye *et al.*, 2001).

References

Boiten E. and Bujorianu M. (2003): *Exploring UML refinement through unification*, In: Critcal Systems Development with UML, Proc. UML'03, (J. Jürjens, B. Rumpe, R. France, and E.B. Fernandez, Eds.). — Technische Universität München, No. TUM-I0323, pp. 47–62.

Cengarle M.V. and Knapp A. (2004): *UML 2.0 interactions: Semantics and refinement*, In: Critical Systems Development with UML, Proc. CSDUML'04 (J. Jürjens, E.B. Fernandez, R. France and B. Rumpe, Eds.). — Technische Universität München, pp. 85–99.

Clark T. (2000): *Object-oriented refinement and proof using behaviour functions*. — Proc. 3rd Workshop *Rigorous Object-Oriented Methods*, York, electronic edition available at: `http://ewic.bcs.org/conferences/2000/objectmethods/papers/paper2.htm`.

Egyed A. (2002): *Automated abstraction of class diagrams*. — ACM Trans. Soft. Eng. Meth., Vol. 11, No. 4, pp. 449–491.

Fowler M. and Scott K. (2000): *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. — Reading, Massachusetts, Addison-Wesley.

Fowler M., Beck K., Brant J., Opdyke W. and Roberts D. (1999): *Refactoring: Improving the Design of Existing Code*. — Reading, Massachusetts, Addison-Wesley.

Harel D. and Politi M. (1998): *Modeling Reactive Systems with Statecharts: The Statemate Approach*. — New York: McGraw-Hill.

Hnatkowska B., Huzar Z., Kuźniarz L. and Tuzinkiewicz L. (2003): *Refinement relationship between collaborations*. — Proc. 2nd Workshop *Consistency Problems in UML-based Software Development* (in conjunction with UML03), San Francisco, pp. 51–57.

Hnatkowska B., Huzar Z. and Tuzinkiewicz L. (2004a): *Collaboration and class diagram consistency*. — Found. Comput. Dec. Sci., Vol. 29, No. 1–2, pp. 75–89.

Hnatkowska B., Huzar Z. and Tuzinkiewicz L. (2004b): *Structure refinement of class diagrams*. — Proc. Conf. *Information Systems Implementation and Modelling*, Roznov pod Radhostem, Czech Republik, pp. 37–44.

Hnatkowska B., Huzar Z. and Tuzinkiewicz L. (2004c): *On understanding of refinement relationship*. — Proc. 3rd Workshop *Consistency Problems in UML-based Software Development* (in conjunction with UML04), Lisbon, Portugal, pp. 11–22.

Hubert R. (2001): *Convergent Architecture. Building Model-Driven J2EE Systems with UML*. — New York: Wiley.

Jacobson I., Booch G. and Rumbaugh J. (1999): *The Unified Software Development Process*. — Reading, Massachusetts: Addison-Wesley.

Katara M. and Mikkonen T. (2002): *Refinements and aspects in UML*. — Proc. Workshop *Aspect-Oriented Modeling* (in Conjunction with UML02), Dresden, Germany, electronic edition available at: `http://lgl.epfl.ch/workshops/uml2002/papers/katara.pdf`.

Knapp A. and Merz S. (2002): *Model checking and code generation for UML state machnines and collaborations*. — Proc. 5th Workshop *Tools for System Design and Verification*, Institut fur Informatik, Universitat Augsburg, pp. 59–64.

Knapp A., Merz S. and Rauch Ch. (2002): *Model checking timed UML state machines and collaborations*. — Proc. 7th Int. Symp. *Formal Techniques in Real-Time and Fault Tolerant Systems*, Berlin, Germany, pp. 395–416.

Kruchten P. (1999): *The Rational Unified Process. An Introduction*. — Reading, Massachusetts: Addison-Wesley, Longman.

Lano K. and Bicarregui J. (1999): *Semantics and transformations for UML models*. — Proc. 1st Int. Workshop *Unified Modeling Language*, (in conjunction with UML'98: Beyond the Notation), Mulhouse, France, pp. 107–119.

Liu Z., Li X, Liu J. and Jifeng H. (2004a): *Integrating and Refining UML Models*. — UNU/IIST Report, No. 295.

Liu Z., Jifeng H., Li X. and Chen Y. (2004b): *Consistency and Refinement of UML Models*. — Proc. 3rd Int. Workshop *Consistency Problems in UML-based Software Development* (in conjunction with UML04), Lisbon, Portugal, pp. 23–40.

Manassis E. (2004): *Practical Software Engineering. Analysis and Design for. NET Platform*. — Boston: Addison-Wesley.

OMG (2003): *OMG Unified Modeling Language: Superstructure, Version 2.0*. — Final Adopted Specification.

Pratt V. (1986): *Modelling concurrency with partial orders*. — Int. J. Parallel Program., Vol. 15, No. 1, pp. 33–71.

Pons C. and Kutche R.D. (2004): *Traceability across refinement steps in UML Modelling*. — Proc. of 3rd Workshop *Software Model Engineering, WiSME 2004*, Satellite Workshop at the 7-th Int. Conf. *UML*, Lisbon, Portugal, available at: `http://www.metamodel.con/wisme-2004/present/19pdf`

Pons C., Giandini R. and Baum G. (2000): *Dependency relations between models in the Unified Process*. — Proc. IEEE Int. Workshop *Software Specification and Design IWSSD*, San Diego, CA, USA, pp. 149–157.

Rumbaugh J., Jacobson I. and Booch G. (2004): *The Unified Modeling Language. Reference Manual, 2nd Ed.* — Reading, Massachusetts: Addison Wesley.

Souza D.F.D. and Wills A.C. (1999): *Objects, Components and Frameworks with UML – The Catalysis Approach*. — Reading, Massachusetts: Addison-Wesley, Longman.

Sunye G., Pollet D., Traon Y. and Fezequel J.(2001): *Refactoring UML Models*, In: Modeling Languages, Concepts, and Tools, Proc. UML01 (M. Gogolla, C. Kobryn, Eds.). — Berlin: Springer, pp. 134–148.

Received: 24 February 2005
Revised: 27 November 2005