The Role of Synthesis Constraints in Role Modeling

Technical Report
IC/2002/038

Pavel Balabko, Alain Wegmann
Laboratory of Systemic Modeling (IC-LAMS)
Swiss Federal Institute of Technology – Lausanne
EPFL-IC-LAMS
CH-1015 Lausanne, Switzerland
{ pavel.balabko,alain.wegmann}@epfl.ch

Abstract. To reuse existing specifications and increase the speed of development, modern development methods widely use design patterns and collaborations. Both, design patterns and collaborations, use the concept of role as a basic modeling concept. To specify models where one object may play several roles, a synthesis operation (the composition of two base roles in a third role) has to be specified. All role-based approaches have difficulties specifying role synthesis. As a consequence, synthesis is never specified without the description of the actual implementation of the synthesis. To specify synthesis at a higher level of abstraction, independent of implementation, requires the proper understanding of relationships between roles, when they are put together in one common context. In this paper we define the concept of "synthesis constraints" that shows relations between roles. We show how "synthesis constraints" can be used to specify the role synthesis operation. Using "synthesis constraints" allows a designer to make explicit his decisions about how the synthesis is done in an abstract and implementation independent way. Specifying synthesis with "synthesis constraints" is a powerful technique that can be used in many different domains, especially in business engineering. The use of roles allows a developer to specify separately certain concerns of a business system. This enables the discovery of new business models for a business system by means of different disassembling and assembling of roles.

1 Introduction

The modern market requires a rapid development of business systems and IT systems. Modeling of systems can help to reduce the development time. One of the key techniques developers can use when designing a system is to define the collaborations between the system's parts. Building collaborations can be done rapidly by reusing gained experience. This experience can be documented in a form of design patterns [Fowler96, Gamma95]. Specifying collaboration as well as working with design patterns is often related with roles and role modeling.

There is no consensus on the definition of role [Steimann00]. In our work we define a role as an abstraction of the behavior of an object. A role is always defined in a context that is collaboration with other roles [Genilloud00]. The collaboration defines how several roles interact together to achieve a common result. The specification of a collaboration and roles participating in this collaboration is a role model [Reenskaug96]. Fig. 1.a shows two role models. Each role model is associated with two roles.

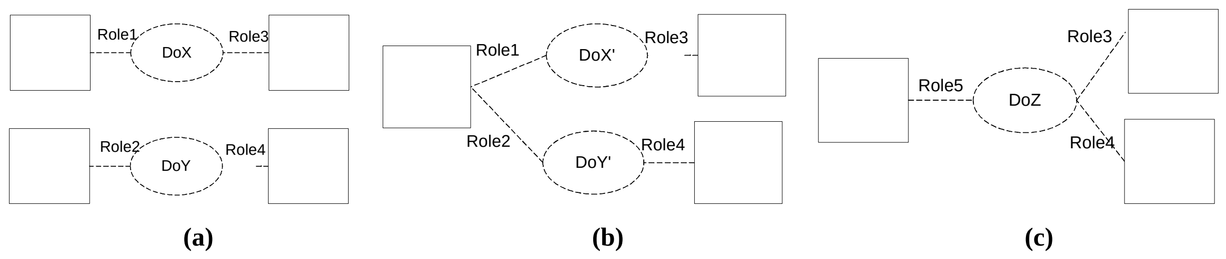


Fig. 1 **(a)** Two base role models: "DoX" and "DoY".
(b) An intuitive approach to specify synthesis of roles.
(c) A new synthesized role model: Role5 is result of the synthesis of Role1 and Role2.

During the development process, the developer needs to combine (or synthesize) roles to make bigger roles. Basic roles are the roles that we want to combine and synthesized roles are these that are the result of a combination (or a synthesis) of basic roles. The specification of the synthesis of basic roles is an important but not evident task. It has been discussed in many works [Reenskaug96], [Riehle97], [Nguyen00], [Kendall00]. Still there is no consensus on how the specification of the synthesis can be done; and often, misleading synthesis specifications are done based on developer's intuition. Let's give an example. Intuitively, when we have to specify the behavior of an object that plays two mutually dependant roles (Role1, Role2 in Fig. 1.a) defined in different role models, we specify that this object plays both roles (Role1, Role2 in Fig. 1.b). Saying that, we ignore the fact that these roles are mutually dependant, and that the behavior of each role is influenced by the behavior of other role. It would be more correct to say that an object plays another new role that has been defined as a synthesis of the original roles (Role5 in Fig 1.c). So we can conclude that Fig. 1.b is incorrect and that the correct specification of a synthesized role model should be as it is shown in Fig. 1.c: each role has only one association with the collaboration where this role participates. But this figure does not explain the way the synthesized role (Role5) was built from the set of original roles (Role1 and Role2). So our goal is to show how the synthesis of base role models is done.

In this work we define a new concept, called *synthesis constraints,* which allows a developer to specify a synthesis of base role models. To define synthesis constraints, in section 2 we give detailed definitions of role and role model. In section 3 we discuss issues related with the synthesis of role models: first we consider the semantics of role model synthesis and define synthesis constraints based on the semantics. Then we consider how the synthesis of role models can be specified using role synthesis specification. Role synthesis specification includes specifications of base roles and synthesis constraints relating these specifications of base roles. In the end of the section 3 we show how a role synthesis specification can be implemented for different infrastructures; we consider the enterprise infrastructure, the OO infrastructure and the constraint base infrastructure. In section 4 we show the practical impact of the synthesis constraints. We show how synthesis constraints allow a developer to explicit his design decisions made in the synthesis of role models. For this purpose we consider an example in the field of business engineering. This example clearly demonstrates the power of the new approach proposed in this paper. Section 5 is the conclusion

2 Role and Role Model Definitions

In order to consider the synthesis of role models we have to choose a consistent semantic framework. We use the ISO/ITU standard "Reference Model for Open Distributed Processing" – part 2[1] [ISO96] as a framework.

Based on RM-ODP, any modeling activity consists of identifying entities in the Universe of Discourse. The **universe of discourse** corresponds to what is perceived as being reality by a developer and **entity** is "any concrete or abstract thing of interest" [clause 6.1] in the universe of discourse. Identified entities are modeled as **model elements** in a **model**. Model elements can be different modeling concepts (object, action, behavior etc). In this section we give the definitions of some modeling concepts that are necessary for the understanding of our paper[2]. We start with the definition of the object concept. If in the universe of discourse we have entities that can be modeled with state and behavior, we model this entity as objects:

Object: *"An object is characterized by its behavior and dually by its state"* [clause 8.1].

Let's note that the duality of state and behavior means that the state of an object determines the subsequent behavior of this object. The definition of an object is based on the definition of behavior and state:

Behavior: *A collection of actions and a set of (sequential) relations between actions.*
State: *A collection of attributes, attribute values and a set of relations between attributes.*

During a system evolution, attributes can change their values; relations between attributes can also change. To specify these changes we will use pre and postconditions. We will also need the concept of time to specify the ordering of actions. For any action we will specify "*an interval of arbitrary size in time at which action can occur"* **[**clause 8.10**]**. We group time intervals into the partially ordered set of time points. This set of time points defines sequential relations between actions. Thus the behavior of an object can be fully described as a tuple: J = <*A, SeqRels, T, AVals, Attrs, AttrRels, Pre, Post, instant_begin, instant_end, precondition, postcondition* >, where each element of this tuple can be a set of elements or a function:

$A = \{a_1, a_2, \ldots\}$	is a set of actions
SeqRels: *A,A®{true, false}*	is a total function[3] that defines sequential relations between actions
$T = \{t_1, t_2, \ldots\}$	is a partially ordered set of time points
$AVals = \{val_1, val_2, \ldots\}$	is a set of attribute values

[1] Later in our work we will reference RM-ODP – part 2 by indicating the corresponding clause in square brackets.
[2] Other definitions you can find directly in the RM-ODP standard.
[3] Total function is a function that is defined for each value of the input set.

Attrs={attr$_1$,attr$_2$,…}, where *attr$_n$: T®AVals*	is a set of attributes, where each attribute is a total function that returns the attribute value for a given time
AttrRels={rel$_1$,rel$_2$,…}, where *rel$_n$: T® Attrs, Attrs*	is a set of relations between attributes, where each relation is a function that returns two attributes for a given time
Pre={pre$_1$,pre$_2$,…}	is a set of preconditions (predicates defined on the set of attributes and set of values)
Post={post$_1$,post$_2$,…}	is a set of post-conditions (predicates defined on the set of attributes and set of values)
instant_begin: A®T	is a total function that returns the instant in time when the action starts
instant_end: A®T	is a total function that returns the instant in time when the action ends
precondition: A®Pre	is a total function that returns a precondition predicate for a given action
postcondition: A®Post	is a total function that returns a postcondition predicate for a given action

Now we can consider the definition of the concept of a role[4]:

Role: ***"****An abstraction of the behavior of an object." A role is always defined in a context that is collaboration with other roles.*

Since a role is an abstraction of the behavior of an object, we can model a role with a tuple R (similar to ϑ), where sets of actions, constraints, time points attributes etc are limited for a given context (a given collaboration):

$$R = abstaction(context, J)$$

To represent the context where a role is defined in this work we will use the *role model* term, inspired by OOram [Reenskaug96]:

Role Model *specifies the structure of collaborating roles (RoleModel = {R$_1$, R$_2$,…,R$_n$}) along with their state and dynamic properties:*
R_n = *<A_n, $SeqRels_n$, T_n, $AVals_n$, $Attrs_n$, $AttrRels_n$, Pre_n, $Post_n$, $instant\_begin_n$, $instant\_end_n$, $precondition_n$, $postcondition_n$ >*

To reference elements in the tuple R_n we will use the following notation: $R_n.Set_x.Element_y$, where Set_x can be any set from the tuple R_n; or $R_n.function_x(\ldots)$, where $function_x$ can be any function from the tuple $R_{n.}$ For example, $R_1.precondition(R_1.A.a_1)$ is a precondition for an actions $a_1$in the specification of the role R_1.

To represent visually the structure of collaborating objects with their state and dynamic properties, we use the notation inspired by UML [OMG99]. We represent a role model (see Fig. 2) by a collaboration (a dashed oval), role specifications (boxes) and role names (names on the line connecting an oval with a box). This notation is similar with the notation used in UML collaboration diagram. The difference is that instead of an attribute compartment in UML (middle pane in each box) we use graphical notation inspired by UML class diagram to represent a state as a set of attributes and relations between them. Instead of the compartment that holds a list of operations in UML (lower pane in each box) we use the graphical notation inspired by UML activity diagram to represent a role behavior.

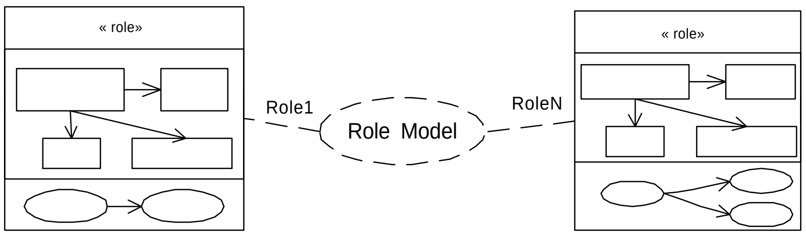


Fig. 2. Graphical representation of a role model

3 Synthesis of Role Models

In this section we define synthesis constraints and consider how synthesis constraints can be used by a developer in a design process: in subsection 3.1 we give the definition of synthesis and explain its semantics; in subsection 3.2 we define synthesis constraints and show how synthesis constraints can be used to model synthesis; in subsection 3.3 we show the implementation of synthesis constraints.

[4] This definition is inspired by the definition of Role in RM-ODP and [Genilloud00]. We use our own definition because we believe that it is clearer for people who do not have a deep knowledge of RM-ODP.

3.1 Synthesis

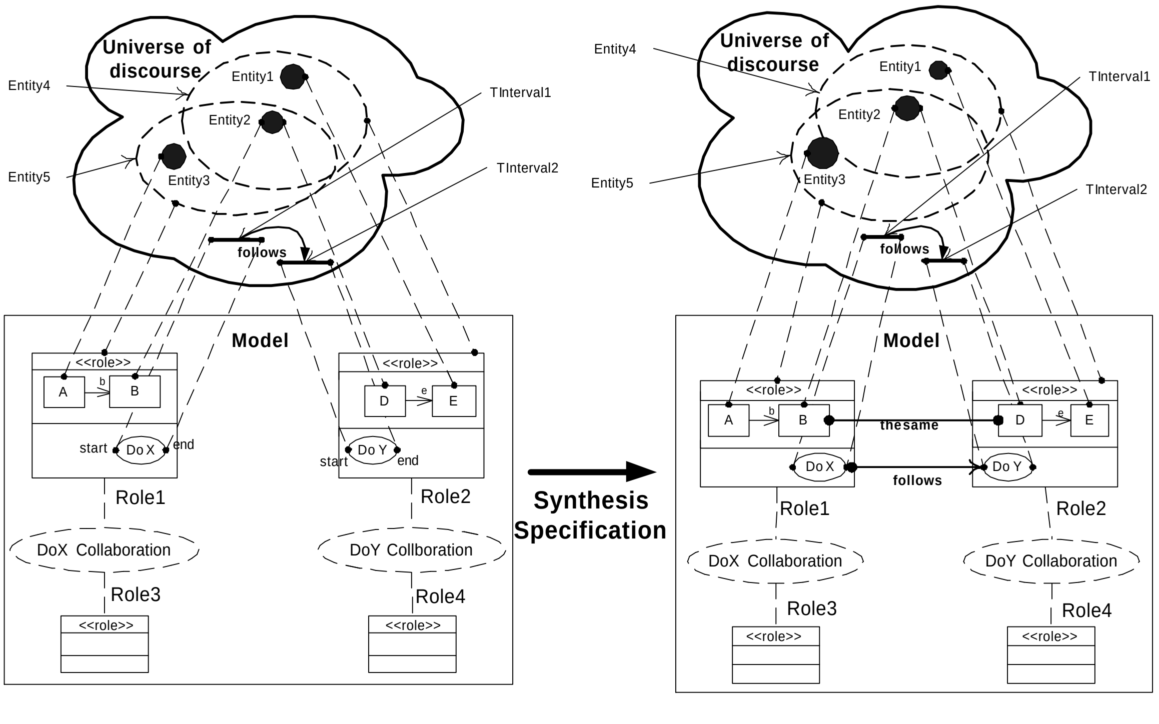
Working with roles allows a developer to concentrate on a certain concern of a system that is modeled as a role model. But at some point a developer needs to take into consideration other role models that exist. To take into consideration other role models, we have to consider a synthesis operation. We took the definition of the synthesis from [AllWords99]:

Synthesis ***(AllWords.com):*** *1. A complex whole formed by combining; 2. The combining of the constituent elements of separate material or abstract entities into a single or unified entity.*

This definition considers two meanings of synthesis: the first meaning describes synthesis as a result of the combination process. The second one defines the process of combination that explains how the combination was done. Let's consider the synthesis of role models. Accordingly to the definition of synthesis, role model synthesis should specify:

- The result of the combination of base role models.
- How the combination of base role[5] models was done;

The first specification (the result of the combination) can be done with a new synthesized role model (Fig.1.c) as we have shown in section 1. In order to understand how to make the second specification (showing how the combination was done), we start with semantics of the role model synthesis. We discuss it with the example shown in Fig. 3.



(a) **(b)**

Fig. 3. The semantics of synthesis constraints; a: Two base role models and their semantic meaning; b: Role synthesis specification: base role models and synthesis constraints

The upper part of Fig.3 shows the universe of discourse. The lower part of Fig.3 represents the model of the universe of discourse. Let's suppose that we have a developer who modeled separately two concerns of a system with two role models: the DoX role model and the DoY role model. We can see that in the DoX role model the developer decided to model Entity5 as Role1 (where he modeled Entity2, Entity3 as attributes of Role1 and TInterval1 as an interval where the action DoX takes place). In the DoY role model he decided to model Entity4 as Role2 (where he modeled Entity1, Entity2 as attributes of Role2 and TInterval2 as an interval where the action DoY takes place). To make a bigger model he wants to synthesize these two role models. If we consider the semantics of role synthesis we can see that roles can influence each other, i.e. behavior of different roles can influence behavior of other roles involved in synthesis. By looking in the universe of discourse we can see that the attribute B of Role1 and the attribute D of Role2 have the same semantic meaning since

[5] We call role models before synthesis *base role models* and correspondingly roles, *base roles*. And we call role models after synthesis *synthesized role model* and correspondingly roles, *synthesized roles*.

they model the same Entity2. This means that in the specification of role synthesis we have to be able to express the semantic equality of role attributes. Another thing we can see is that the time interval of the action DoY follows the time interval of the action DoX. To preserve the sequence in the model we require that the action DoX happens before the action DoY.

The mutual influence between roles can be specified in several ways. Examples of different syntax for expressing role's mutual influence can be seen in [Riehle98], [Fiadeiro01], [Reenskaug96]. But the syntax proposed in these works depends on implementation details: roles are assigned to some objects and synthesis is specified as a communication through given object interfaces. These technical details complicate our understanding of synthesis semantics. That is why we believe that the syntax abstract from the implementation details can be extremely useful in the modeling of business and IT systems. This can help a developer to concentrate his efforts on design decisions. Decisions taken by a developer have to be based on the semantics of the synthesis rather than on implementation. In our work we define a way to specify the syntax for the synthesis of role models that will reflect the semantics of the synthesis and abstract from the implementation details.

3.2 Role Model Synthesis

In this section we consider how the synthesis of role models can be specified using synthesis constraints. First we give the definition of synthesis constraints (subsection 3.2.1) and then using synthesis constraints we define the Role Synthesis Specification (subsection 3.2.2).

3.2.1 Synthesis constraints

As we have seen in the previous section, we have to define a new concept that will specify how base roles influence each other within a new synthesized model. We use synthesis constraints for this purpose:

Synthesis constraints: *Constraints implied on the behavior of base roles from different base role models.*

In general, synthesis constraints can be specified as a mapping of corresponding elements used in the definition of the basic roles. Let's R_1 and R_2 be two basic roles. In this case synthesis constraints can be defined as a set of couples that show how mapping is done:

$$Synthesis\ constraints = \{c_1, c_2, ...\},\ where$$

$$c_n \overset{def}{=} (R_1.Set_x.element_n \leftrightarrow R_1.Set_x.element_m) \quad or$$

$$c_n \overset{def}{=} (R_1.function_y(R_1.Set_x.element_n) \leftrightarrow R_1.function_y(R_1.Set_x.element_m))$$

where Set_x can be one of the following sets: *A, T, AVals, Attrs, AttrRels, Pre or Post* and $function_y$ can be one of the following functions: *instant_begin, instant_end, precondition, postcondition.*

Note that while defining synthesis constraints we have to ensure the consistency of a synthesized model. For example, if we want to specify mapping of two actions from two different roles, we also have to specify that the preconditions and the postconditions for these actions are the same. In the general case, consistency rule can be specified in the following way: if two elements ($R_1.Set_x.e_n$ *and* $R_2.Set_x.e_m$) from base roles (R_1 and R_2) are mapped to each other (e.g. role1.A.action1 is the same as role2.A.action2) and there is a function (e.g. precondition) in the definition of base roles using these elements as arguments (e.g. precondition(action1)), then the resulting elements of these functions should also be included in the synthesis constraints (e.g. role1.precondition(action1) same as role2.precondition(action2)). So the consistency rule can be described as:

$$(c_n = (R_1.Set_x.e_n \text{ « } R_2.Set_x.e_m))\ \text{Ú}\ (R_1.f(R_1.Set_x.e_n) = R_1.Set_y.e_p)\ \text{Ú}\ (R_2.f(R_2.Set_x.e_m) = R_2.Set_y.e_q)$$

$$\text{Þ} \quad c_{n+1} = (R_1.Set_y.e_p \text{ « } R_2.Set_y.e_q).$$

Below we give definitions of four different synthesis constraints that we have used in our research work[6]. These synthesis constraints respect the consistency rule shown above:

- **Attribute Equality** (*Role$_1$.Attrs.attribute1* •–• *Role$_1$.Attrs.attribute2*)

  $\forall t1 \in Role_1.T, \forall t2 \in Role_2.T:$

  $(Role_1.Attrs.attribute1(t1) \leftrightarrow Role_2.Attrs.attribute2(t2))$

  This means that values of attribute1 and attribute2 should be equal at any time moments specified by Role1 and Role2 correspondingly.

[6] This lest is not exhaustive and can be extended with other useful synthesis constraints.

- **Synchronized Actions** (*$Role_1$.A.action1 ®·¬ $Role_2$.A.action2*).
  $Role_1.instant\_begin(Role_1.A.action1) \leftrightarrow Role_2.instant\_begin(Role_2.A.action2)$
  $Role_1.instant\_end(Role_1.A.action1) \leftrightarrow Role_2.instant\_end(Role_2.A.action2)$
  This means that the actions of each role start and finish at the same time.
- **Constraints of Sequentiality** (*$Role_1$.A.action1 ® $Role_2$.A.action2*).
  $Role_2.instant\_begin(Role_2.A.action2) \leftrightarrow t):$
  $t \in followingTE(Role_1.instant\_end(Role_1.A.action1))$
  This means that the second action can start at any time after the completion of the first action. In details about constraints of sequentiality you can read in [Balabko01].
- **Action Equality** (*$Role_1$.A.action1 ●–● $Role_2$.A.action2*).
  $Role_1.A.action1 \leftrightarrow Role_2.A.action2$
  $Role_1.instant\_begin(Role_1.A.action1) \leftrightarrow Role_2.instant\_begin(Role_1.A.action2)$
  $Role_1.instant\_end(Role_1.A.action1) \leftrightarrow Role_2.instant\_end(Role_2.A.action2)$
  $Role_1.precondition(Role_1.A.action1) \leftrightarrow Role_2.precondition(Role_2.A.action2)$
  $Role_1.postcondition(Role_1.A.action1) \leftrightarrow Role_2.postcondition(Role_2.A.action2)$

  These synthesis constraints specify two actions that have the same results (and thus can be considered identical), i.e. that both roles specify actions with the same pre- and post-conditions and happening at the same time. For example, "Identical actions" can specify that both roles move the same entity in the universe of discourse from one place to another and these moves happen in the same time interval.

3.2.2 Role Synthesis Specification

To specify the synthesis of role models we have to specify a set of base role models and synthesis constraints that relate them. We will call such specification as *Role Synthesis Specification*. Let's consider an example in Fig. 4 that shows two base role models "DoX Model" and "DoY Model".

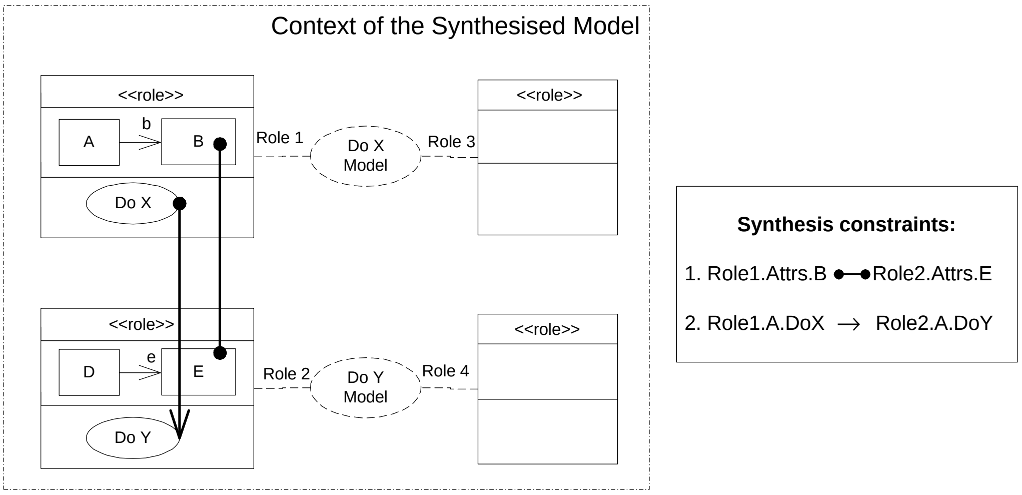


Fig. 4. Role Synthesis Specification

These two base role models are related with two synthesis constraints: "*Role1.Attrs.B ●–● Role2.Attrs.E*" (attribute equality), and *"Role1.A.DoX ®Role2.A.DoY"* (constraints of sequentiality).

The Role Synthesis Specification allows a developer to make explicit his decisions about how synthesis is done. These decisions are made based on what a developer observes in the universe of discourse. The main advantage of the Role Synthesis Specification is that it allows the specification to be abstract from an implementation: at this point a developer does not need to decide on how and by which object the synthesis constraints would be implemented. The designer can use different synthesis constraints to express different ways of synthesizing.

3.3 Role Synthesis Implementation

This section explains how Role Synthesis Specification can be implemented. First we show two major implementation patterns for any underlying technology (subsection 3.3.1). Then we concentrate on the details of implementations for some technologies (subsection 3.3.2).

3.3.1 Two Implementation patterns

In order to move forward to the implementation, we have to take decisions on objects that would be responsible for the implementation of synthesis constraints. Here, two situations are possible: when one object plays both base roles and the same object is responsible for the implementation of the synthesis constraints. In the second situation there are two objects playing base roles and the responsibility for the implementation of synthesis constraints is distributed between them. This brings us to two possible implementation patterns of the Role Synthesis Specification:

- By merging base roles into one synthesized role,
- By extending the behavior of the base roles,

Let's consider an example that shows how the Role Synthesis Specification from Fig. 4 can be elaborated using two implementation patterns. Here we only show the implementation of the "Constraints of Sequentiality" (see Fig. 4.a). The implementation of the "Attribute Equality" synthesis constraints can be specified in a similar way.

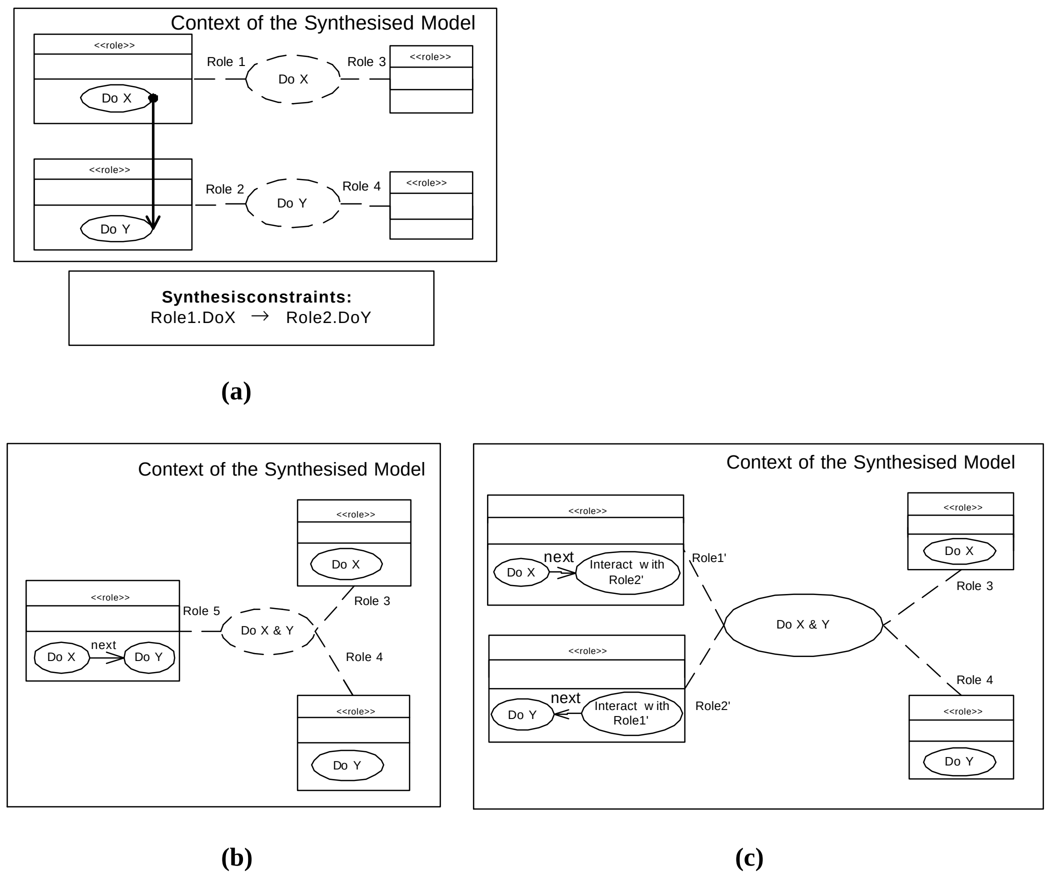


Fig. 5. Applying two implementation patterns for Role Synthesis Specification
(a) Initial Role Synthesis Specification (base role models plus synthesis constraints); possible implementations: (b) Synthesized model: Merging Base Roles and (c) Synthesized model: Extending Base Roles

The first implementation, "*Merging Base Roles*", merges base roles by putting them in the common context of a new role (Role5 in Fig. 5.b). In this case Role5 is entirely responsible for carrying out the synthesis constraints[7]. However this way of specification seems easy, but it has a significant drawback: it does not allow us to separate base roles to implement them with different objects (to keep them separate, for example for the purpose of distribution).

[7] We discuss how a role can be implemented with objects in a given infrastructure in the next subsection.

The second implementation, “*Extending Base Roles*”, specifies synthesis constraints by means of extending the behavior of base roles with some additional behavior that specifies how the base roles can interact (see Fig. 5.c). These additional behaviors are the two actions “Interact with Role1´”, “Interact with Role2´” which are added to Role1 and Role2. These actions allow for communication between roles: the “Interact with Role2´” action in Role1´ specifies sending of the message to Role2´ and the “Interact with Role1´” action in Role2´ allows for the action “DoY” to be executed on receipt of the message. Using the “Extending Base Roles” specification allows us to distribute base roles (Role1 and Role2) between two objects.

Let’s note that two elaborations of Role Synthesis Specification that we did based on implementation patterns are still specified at a high level of abstraction: the implementation is not yet well defined. The implementation depends on the infrastructure that a developer uses. In the next subsection, we consider several infrastructures and explain how the implementation can be done for any of them.

3.3.2 Practical Role Synthesis Implementations

The implementation of the synthesis constraints depends on the capacity of the computational objects[8] that a certain infrastructure or technology can provide. In the context of design of business and IT systems, we considered three types of object infrastructures where the implementation of the synthesis constraints is especially interesting: enterprise infrastructure, the OO infrastructure, constraint base infrastructure.

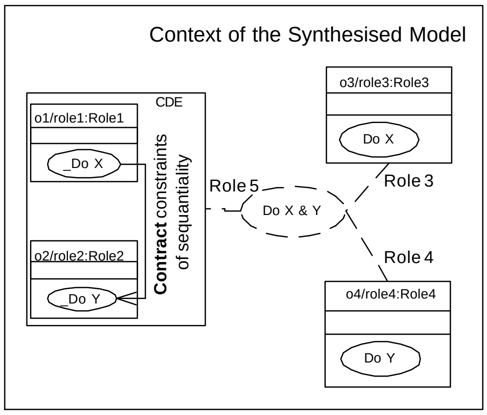
Enterprise infrastructure: The computational objects are human beings (or enterprise units as departments). A role model can be implemented in the form of guidelines for a group of people that are trying to achieve some common goal in a given activity. Synthesis constraints in this case can be implemented as prescriptions that explain how different activities, where a person may participate in, should be constrained. For example, we can imagine the following prescription for a person playing Role5 in the case of the “Merging Base Roles” specification: “First, you have to finish your DoX work with a person that plays Role 3, and then you have to do your DoY work with a person that plays Role4”. In the case of “Extending Base Roles” the prescription for a person playing Role1´ would be: “Finish your DoX work and immediately inform a person playing Role1´”. The prescription for a person playing Role2´ would be: “Wait until a person playing Role1´ finishes his work, and then do your DoY work”.

OO infrastructure (like Java): The computational objects in OO programming languages are instances of classes. In the design phase, a class of a programming language can be assigned to play several roles from one or more role models (or collaborations). In this case, synthesis constraints allow a designer to make explicit design decisions about how different role types played by a class are synthesized. Synthesis constraints are implemented by a programmer who codes methods implementing constraints. Inside a class, a programmer has full accesses to this class methods and attributes, thus implementation of synthesis constraints is straightforward. Once the implementation of the synthesis constraints has been programmed, the behavior of classes can neither be changed nor constrained. This means that an object in pure OO infrastructure cannot acquire or abandon roles dynamically. To have this capability we have to use another infrastructure.

Constraint based infrastructure: It is based on the idea of coordination contracts proposed in [Fiadeiro01]. In this infrastructure there are two kinds of computational objects: Java objects that can play base roles from the synthesized role models (for example Role1 and Role2 in Fig. 5.a), and Coordination Development Environment[9] (CDE) that can constrain the behavior of base roles based on a coordination contract. “In general terms, a coordination contract is a connection that is established between a group of objects (participants). Through the contract, rules and constraints are superposed on the behavior of the participants.” [CDE 01] The example of such a contract is as shown in Fig. 6. It prohibits the execution of the DoY action of the object o2 (that plays Role2) before the execution of the DoX action of the object o1 (that plays Role1). The implementation of other synthesis constraints can be done in a similar way.

[8] Based on RM-ODP a computational object is an object that is able to perform computations, thus it can play roles.

[9] See www.atxsoftware.com for details about CDE



```
contract ConstraintsOfSequantiality
participants   // contract participants
       role1:Role1;
       role2:Role2;
attributes   // attributes of the contract
        boolean doXisFinished=false;
coordination   // rules
       UpdateRule1: when *->> role1.DoX() && (doXisFinished==true)
         do { System.out.println("This action is currently not allowed");};
       UpdateRule2: when *->> role1.DoX() && (doXisFinished==false)
          do { role1._DoX();}
          after {doXisFinished=true;};
       UpdateRule3: when *->> role2.DoY() && (doXisFinished==false)
          do { System.out.println("This action is currently not allowed"); };
       UpdateRule4: when *->> role2.DoY() && (doXisFinished==true)
          do { role2._DoY(); };
 end contract //ConstraintsOfSequantiality
```

Fig. 6. The implementation of the synthesis constraints using a coordination contract

Let's note that in contrast with the OO infrastructure, the constraint based infrastructure allows for the dynamic transfer of role behavior instances and types, from one object to another. In order to do this, a computational object has to be able to constrain role behaviors dynamically. I.e. if a computational object plays Role1 and at some point in time it is asked to play Role2, and in addition there is a specification of synthesis constraints between Role1 and Role2, then this object should implement synthesis constraints dynamically. Let's suppose that CDE has in its environment the object o1, playing the base role Role1 (Fig 6). When we want CDE to add another object, playing the role Role2 (Fig 6) we have to create the instance of this object and put it in CDE. Furthermore, since synthesis constraints are specified between Role1 and Role2, we have to create an instance of the coordination contract (Constraints of Sequentiality) that will implement these synthesis constraints. In the same way CDE can abandon instances of objects playing roles, in which case CDE has to abandon instances of coordination contracts also.

Another infrastructure where synthesis constraints can be implemented in a similar way is based on the notion of "Composition Filters" used in AOP [Bergmans01].

4 Application

In the previous section, we have seen how synthesis constraints can be specified and implemented using different computational objects. We have seen that specifications with synthesis constraints abstract from the implementation details. This allows a developer to concentrate his efforts on design decisions rather than on an implementation. This can be extremely useful while specifying business systems[10]. A developer may not only specify separately certain concerns of a system and then synthesize in one coherent role, but also discover new business models by means of different disassembling and assembling of roles. Systems can be assembled in different ways from the set of base role models. A system being assembled from the set of base role models will depend on which roles are synthesized and how synthesis constraints are specified.

In this work we give an example that shows how a new business system can be assembled from the set of based roles models using synthesis constraints. We believe that this example can show the "power" and the "beauty" of synthesis constraints. This example is taken from the StreamCom[11] project, whose goal was to study problems and issues related to the commercialization of streamed information, like video, audio and news-feeds. The StreamCom project aimed to define a model for the commercialization of streamed information and to develop a demonstrator application that will allow the commercialization of video streams on a pay-per-use basis over broadcast networks.

The goal of this example is to show how a new business role model can be specified based on the set of base role models. The idea is to synthesize base roles of Purchaser, Seller and Accounting into a new role of Retailer. We start at the point where a developer already has three base role models: "Exchange CD lot against money", "Exchange CD against money" and "Report Activity Financial Statement" (see Fig. 7).

[10] Another domain where synthesis constraints can be useful is related with modeling of patterns. The [Bois02] technical report explains how to use synthesis constraints to specify the synthesis of patterns.
[11] See http://cui.unige.ch/OSG/projects/streamCom/ for details.

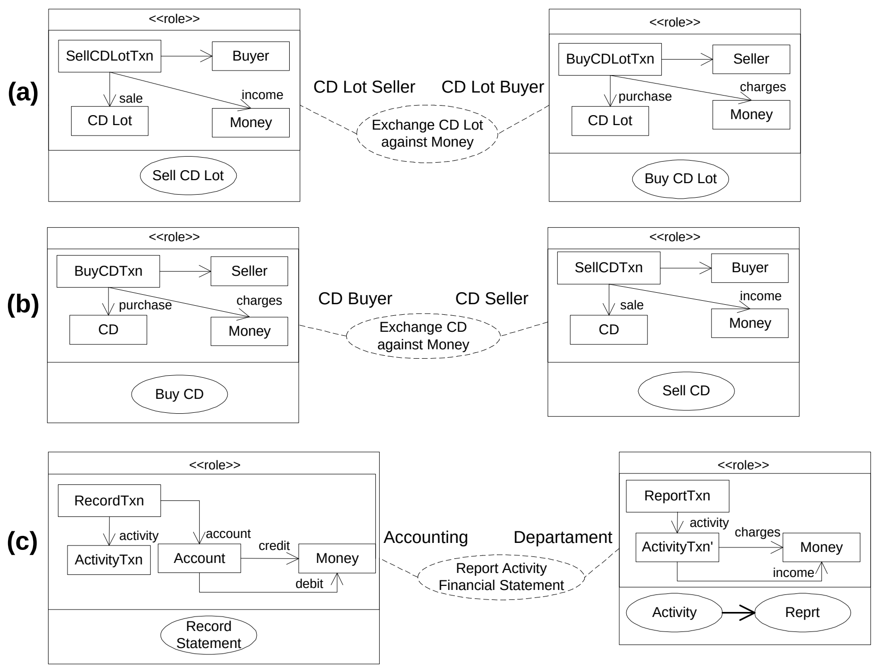


Fig. 7. Three base role models 6.a: “Exchange CD Lot Against Money”, 6.b: “Exchange CD Against Money” and 6.c: “Report Activity Financial Statement”

The first two role models describe CD selling/buying activities. The third model specifies how accounting has to be done for a core company department: first the department accomplishes its core business activity and then sends a financial report about this activity to the accounting department. Then the accounting department credits/debits an amount corresponding to the performed activity to/from the company account.

Based on these three base role models a developer wants to build a new “Retail CD” business role model that will specify CD retailing business. To make this model a developer has to decide on how base roles have to be synthesized. The idea would be to specify the “Retail CD” business role model would be to synthesize the “CD Lot Buyer” role with the “CD Seller” role in such a way that the “CD Seller” and “CD Lot Buyer” will share the same account for their business activities. Following this idea a developer will do a synthesis of base roles in two phases. Decisions made by a developer in each phase will be documented with synthesis constraints.

Phase 1: The goal of the first phase is to include accounting activities in the “Exchange CD lot against money” and “Exchange CD against money” role models. Two role models would be the results of this phase:

“Exchange CD Lot Against Money and Make Financial Report” = Synthesize (
“Exchange CD Lot against money”;
“Report Activity Financial Statement”).

“Exchange CD Against Money and Make Financial Report” = Synthesize(
“Exchange CD against money”;
“Report Activity Financial Statement”).

Phase 2: The goal of the second phase would be to synthesize two resulting role models by means of sharing common accounting activities. We will obtain the following role model:

“Retail CD” = Synthesize (
“Exchange CD Lot Against Money and Make Financial Report”
“Exchange CD Against Money and Make Financial Report”).

In the following two subsections we will consider these two phases in detail.

4.1 Phase 1

As we mentioned above, the goal of the first phase is to include accounting activities in the “Exchange CD lot against Money” and “Exchange CD against Money” role models. To include accounting activities in the “Exchange CD lot against Money” role model we define the following synthesis constraints:

Department.A.Activity ●–● *CDLotBuyer.A.BuyCDLot*
Department.Attrs.ActivityTxn ●–● *CDLotBuyer.Attrs.BuyCDLotTxn*
Department.Attrs.Money ●–● *CDLotBuyer.Attrs.Money*
Department.AttrRels. charges ●–● *CDLotBuyer.AttrsRels.charges*
Accounting.AttrRels.credit = undef (no credit for buying CD Lot activity)
Department.AttrRels.income = undef (no income for buying CD activity)

These two base role models and implied synthesis constraints form the role synthesis specifications (see Fig.7).

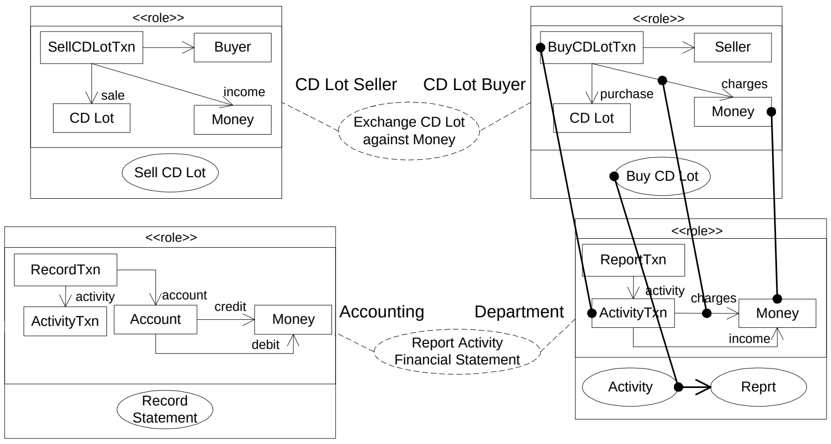


Fig. 8. Role synthesis specification (phase 1)

How the implementation for this synthesized specification will look like? As we said before, we will use only the "merging base roles into one synthesized role" implementation pattern (see Fig. 9):

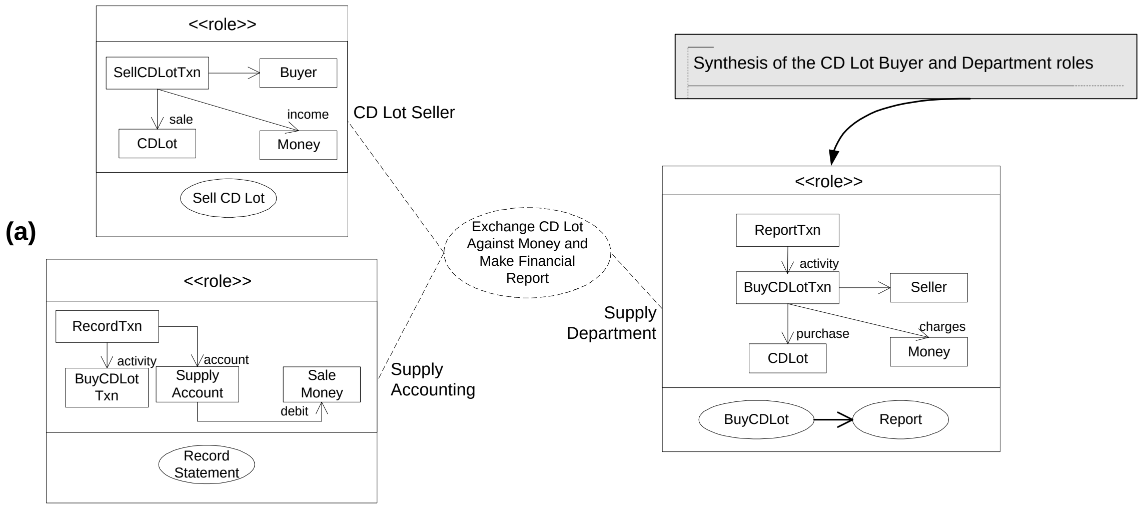


Fig. 9. The implementation of the role synthesis specification (the result of the first phase)

Note, that in the context of the *Supply Accounting* role we have renamed three attributes: *ActivityTxn* into *BuyCDLotTxn; Money* into *SupplyMoney* and *Account* into *SupplyAccount*. New names better show semantics of these attributes.

The role model "*Exchange CD against Money and Make Financial Report*", that includes accounting activities in the "*Exchange CD against Money*" role model, can be obtained exactly in the same way as "*Exchange CD Lot against Money and Make Financial Report*".

4.2 Phase 2

In the second phase of role synthesis we have to specify synthesis constraints in a way that "*Sales Accounting Role*" and "Supply Accounting Role" share the same account and money (see Fig. 10 for the graphical representation of the role synthesis model). We also have to guarantee that the BuyCDLot action precedes the SellCD action:

SupplyAccounting.Attrs.SupplyAccount •–• *SalesAccounting.Attrs.SaleAccount,*
SupplyAccounting.Attrs.SupplyMoney •–• *SalesAccounting.Attrs.SaleMoney*
SupplyDepartment.A.Report ® *SalesDepartment.A.SellCD*

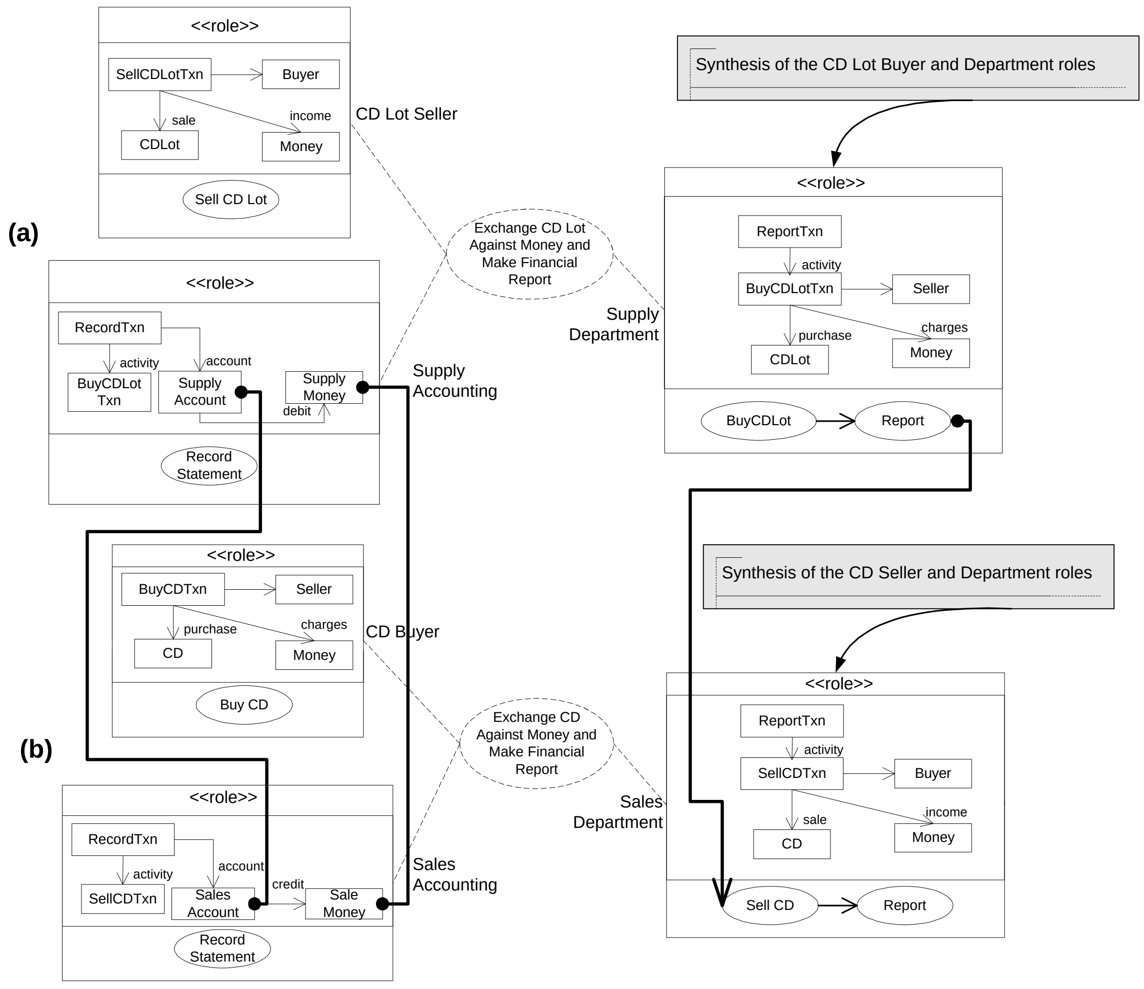


Fig. 10. Role synthesis specification (Phase 2)

Fig. 11 represents the implementation for this role synthesis specification based on the “merging base roles into one synthesized role” implementation pattern.

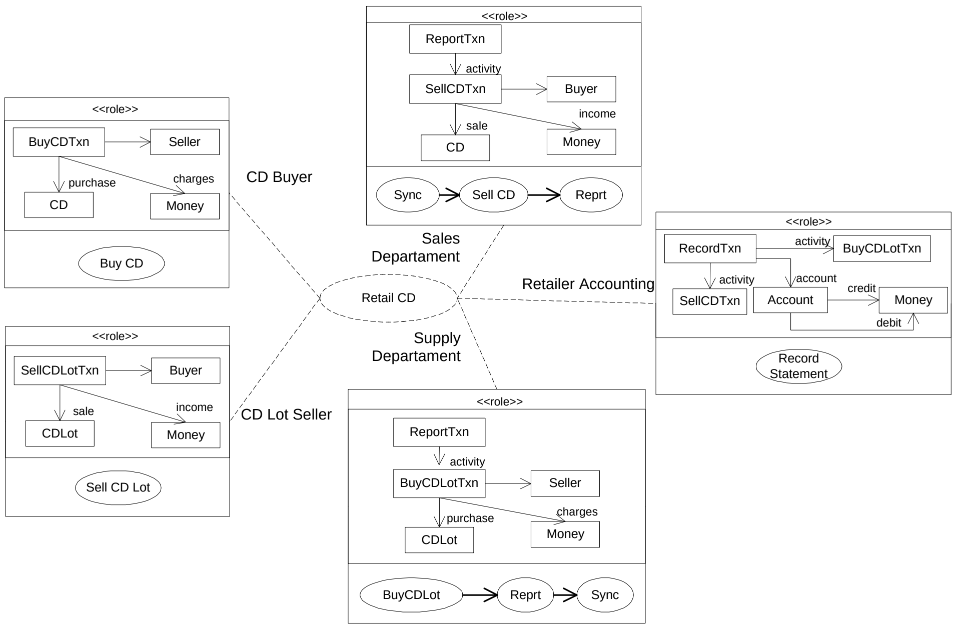


Fig. 11. The "Retail CD" role model (keeping role corresponding to different departments separately)

We can synthesize roles a bit further. We can synthesize the Sales Department, Supply Department and Retailer Accounting roles to get one Retailer role. Synthesis constraints in this case would be that the Money concept should be the same for all three roles. Let's note that by choosing the "Merging Base Roles" implementation we lose the track of how the base roles communicate to each other. Hence we will not be able to separate base roles in order to put them in different execution contexts. However, this implementation of the synthesized role model can be useful for small companies where all functionality is implemented by a single business unit. The final implementation of "Retail CD" is specified in the following way:

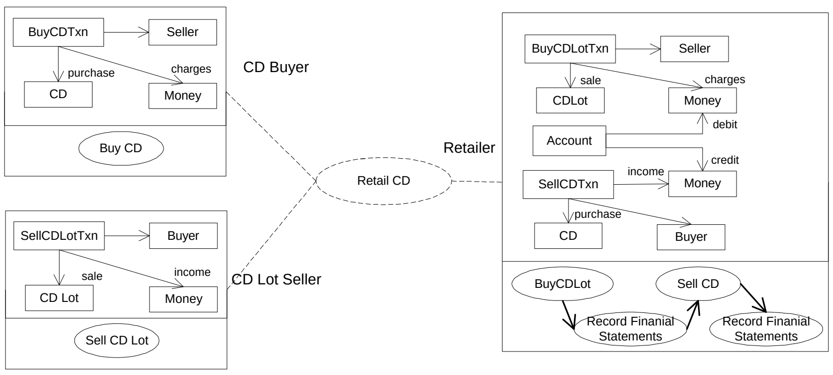


Fig. 12. CD Retailing role model

Let's also note that in order to define the complete "Retail CD" role model, the "CD Seller" and "CD Buyer" roles has to share not only the common account but also a common warehouse for stocking CDs. The modeling of the common warehouse is done exactly the same way and thus we will not show it in our work.

5 Conclusion

In this paper we addressed issues and questions related to the synthesis of role models. We have shown that current approaches for specifying systems do not allow a designer to make explicit his decisions about how the synthesis is done. The main contribution of this work is the definition of a new concept of synthesis constraints that can be used for specifying the synthesis of role models. In this work we have defined four types of synthesis constraints: "State equality", "Synchronized actions", "Constraints of sequentiality" and "Identical actions". We have considered how a model with synthesis constraints can be elaborated for the specification of distributed and non-distributed systems and how these models can be implemented based on different infrastructures.

Synthesis constraints are useful design patterns that can be used for modeling complex role models based on simple ones. Explicit design decisions on the synthesis of base roles, specified with synthesis constraints, allow a developer to assemble and disassemble roles in many different ways and thus create new business and software solutions. We have considered a detailed example in the field of business engineering that shows how base business roles can be synthesized in order to create a new complex business model.

6 Bibliography

[AllWords99] http://allwords.com/ - Copyright © 1998,1999 AllSites.com, Inc.

[Balabko01] Balabko, P.,Wegmann, A., *From RM-ODP to the formal behavior representation*, in *Proceedings of Tenth OOPSLA workshop on Behavioral Semantics*, September 2001

[Bergmans01] Bergmans, L., Aksit, M., *Composing Crosscutting Concerns Using Composition Filters, Communications of the ACM*, Vol. 44, No. 10, pp. 51-57, October 2001.

[Bois02] Bois, F., *Technical report: Specifying patterns in OOAD [in French]*, EPFL-IC-LAMS, 2002, http://icawww/balabko/Projects/StudentProjects/PatternsInOOAD/index.htm

[CDE 01] Coordination Development Environment (CDE) documentation, ATX Software 2001, http://www.atxsoftware.com/.

[Fiadeiro01] Fiadeiro, J., A. Lopes, and M. Wermelinger, *A Mathematical Semantics of Architectural Connectors*, . 2001.

[Fowler96] Fowler, M., *Analysis Patterns: Reusable Object Models*. Addison-Wesley Object Technology: Addison-Wesley Object Technology Series. 1996: Addison Wesley Publishing Company.

[Gamma95] Gamma, E., *et al.*, *Design Patterns*. 1995: Addison Wesley Publishing Company.

[Genilloud00] Genilloud, G. and A. Wegmann. *A Foundation for the Concept of Role in Object Modelling.* in *EDOC*. 2000.

[ISO96] ISO/IEC 10746-1, 3,4 | ITU-T Recommendation X.902, *Open Distributed Processing - Basic Reference Model - Part 2: Foundations* . 1995-1996.

[Kendall00] Kendall, E.A., *Role modeling for agent system analysis, design, and implementation.* IEEE Concurrency, 2000, April-June. **8**(2): p. 34 -41.

[Nguyen00] Nguyen L.T., L. Zhao, and B. Appelbe, *A set approach to role modeling*. in *37th International Conference on Technology of Object-Oriented Languages and Systems*. 2000. TOOLS-Pacific 2000.

[OMG99] OMG, *Unified Modeling Language Specification, v 1.3*, . 1999.

[Reenskaug96] Reenskaug, T., *et al.*, *Working With Objects: The OOram Software Engineering Method*. 1996 ed: Manning Publication Co.

[Riehle97] Riehle, D., *Composite Design Pattern*. in *Conference of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'97)*. 1997: ACM Press.

[Riehle98] Riehle, D. and T. Gross. *Role Model Based Framework Design and Integration*. in *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '98)*. 1998: ACM Press.

[Steimann00] Steimann, F., *On the Representation of Roles in Object-Oriented and Conceptual Modelling.* Data and Knowledge Engineering 35, 2000(1): p. 83–106.