Integrating and Rapid-Prototyping UML Structural and Behavioural Diagrams Using Rewriting Logic

Nasreddine Aoumeur and Gunter Saake

Institut für Technische und Betriebliche Informationssystem
Otto-von-Guericke-Universität Magdeburg
Postfach 4120, D–39016 Magdeburg
{aoumeur,saake}@iti.cs.uni-magdeburg.de

Abstract. Although the diversity of UML diagrams provides users with different views of any complex software under development, in most cases system designers face challenging problems to keeping such diagrams coherently related. In this paper we propose to contribute to the tremendous efforts being undertaken towards rigorous and coherent views of UML-based modelling techniques. In this sense, we propose to integrate most of UML diagrams in a very smooth yet sound way. Moreover, by equipping such integration with an intrinsically concurrent and operational semantics, namely rewriting logic, we also provide validation by rapid-prototyping using Maude implementations.
More precisely, the diagrams we propose to smoothly integrate include: object- and class-diagrams with their related object constraints (using OCL), statecharts and life-cycle diagrams. The integration of such diagrams is based on very appealing Petri-net-like semi-graphical notations. As further advantages of the proposed integration we cite: (1) an explicit distinction between local features and observed ones in (the enriched) class-diagrams which offers a clean separation between intra- and inter-class-diagram reasoning; and (2) a full exploitation of rewriting logic reflection capabilities for expressing different object-life cycles in a runtime way.

1 Introduction

Standardized by the Object Management Group (OMG) in 1997, the Unified Modeling Language (UML) [BJR98, BJR97] methodology has been rapidly accepted and emerged as a suitable framework for modeling (and implementing) complex software-intensive systems. By providing numerous forms of very appealing semi-graphical diagrams with associated texts (i.e. using the object constraint language OCL), UML has been largely experienced in different categories of software-intensive systems. However, as designers attempt to go beyond the syntactical constructions of such diagrams—including object-, class-, sequence-, state-chart-, collaboration-, and component-diagrams with associated

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 296–310, 2002.

text descriptions—they face challenging problems in keeping such diagrams coherent and intrinsically related. Such coherence is a crucial requirements for ensuring consistency and completeness (using different verification/validation formal techniques) of the whole system before its implementation.

As a result of this unsatisfactory state of affair, several proposals have been forwarded recently aiming at bringing more rigor and coherence to these often redundant and incoherent views. Among these proposals we specifically cite the development of an adequate integration, denoted by `Casl-Ltl` [RCA01, ACR00], of the recently developed algebraic specification language `Casl` [Mos97] and a suitable form of labelled transition systems [Ast99]. Using this integration the authors show how almost all UML diagrams can find a rigorous formalization. Other approaches concentrating on the formalization and integration of some UML diagrams have been also put forward like the formalization using Object-Z, Graph-theory, Petri nets, etc (see the proceedings deserved to this methodology [FR99, EK00]).

The purpose of this paper fits within the direction of these research directions, and introduces a more coupled integration of most UML diagrams having in mind complex distributed information systems as a main application domain. That is, following our experiences in this field [AS99c, Aou00, JSHS96, CRSS98], we are concentrating more on object-, class- (with related text descriptions using OCL), transition- and statecharts' diagrams, that are in our view largely sufficient for covering most of structural as well as behavioural aspects in complex information systems. However, instead of describing (or after describing[1]) them separately when conceiving a complex information system, we rather propose to soundly integrate them in a smooth way keeping all their expressive advantages while overcoming most of their shortcomings. More precisely, the shortcomings we are tackling with—as triggers towards the proposed integration—include the following:

- By independently conceiving object constraints—particularly pre-, post-conditions and conditions to be associated with methods or operations in the class-diagrams—, in our view this does not only violate the *intrinsic dependency* of these constraints to associated operations and objects but also increases the degree of incoherence between the two parts, which in fact concern the same world entities. So, our contribution aims at intrinsically incorporating these constraints in the corresponding class- and/or object-diagrams.
- UML diagrams promote just a community-based perception of the system, whereas to cope with the ever-increasing complexity in real-life information systems rather a *component*-based perception is overwhelmingly needed. In this sense, an explicit distinction between local attributes / operations and observed ones, would allow each (hierarchy) of class-diagram—capturing an independent part of whole system— to be autonomously conceived as a

---

[1] We should point out here that the present proposal should be regarded just as complement artifacts helping the UML-based designers for more reliability rather than as a new alternative.

component. On the basis of observed features, such components may be then interconnected by hiding all their internal features.
- Although the object-orientation with its message-passing concept promotes true-concurrency and distribution, the UML (behavioural) diagrams offer only a very restricted form of interleaving (see [WMB99] for recent attempts to deal with concurrency in UML state-charts). That is, a true concurrent semantics would very be helpful for capturing the distributed nature of complex information systems.
- In the same spirit as for OCL descriptions against object and class-diagrams, we also argue that the modelling of life-cycle- and sequence-diagrams independently of class- and object-diagrams makes very hard the understanding and the coherence of whole specification as well as any further refinement steps towards efficient implementations.

In some detail, with the aim to overcome the above UML shortcomings the integration we are proposing may be sketched as follows. But before we should once again clearly point out that our integration is to be regarded as an *intermediate* phase between the UML modelling and implementation phases. The purpose of this intermediate phase, that could be generated (semi-)automatically from the original UML diagrams, is to bring more coherence, concurrency, more componentization and validation to the modelled system.

- First as we just mentioned, in any class-diagram we make an explicit distinction between local attributes / operations[2] and observed ones. Of course such distinction is intrinsically depending on the application at hand. Second, besides the attribute identifiers and their sorts (and eventually initial values), we propose to endow each attribute with a *variable*(s), which will play the role of a *current value* when we proceed to its interpretation using rewrite logic. In the same way, we equip each message argument with a corresponding variable.
- The second important step consists in constructing the dynamic of each message or method-invocation. To this aim we propose to construct for each local message a Petri-net-like 'transition', where the condition and post-operation or the resulting change have to be adapted from the corresponding OCL description when it exists; otherwise they have to be constructed from the intuitive meaning of such a message. A general pattern of such a dynamics is proposed. We will refer to such 'enriched by dynamics' class-diagrams as enriched class-diagrams or simply as components.
- With respect to such a general pattern, we propose to interpret the operations dynamics in terms of rewrite logic. That is, each operation or message dynamics is captured by a corresponding rewrite rule. By allowing objects to be created and deleted, using these rules we show how a true concurrent reasoning is possible with a full exhibition of intra- and inter-object concurrency using an adequate extension of MAUDE language that we have proposed in [AS99a].

---

[2] In order to emphasize the concurrent character of our integration, we will use later messages instead of operations or method-invocations.

- After associating with each class-diagram its corresponding local behaviour, the next step is to deal with the interconnection of different independent (i.e. related only through relationships) class-diagrams composing the system. We follow the same reasoning as for the internal behaviour. That is, for each message declared as observed in each class-diagram as well as for each (dynamical) relationship, we construct the corresponding dynamics using the same Petri-like notation, but at this level only observed features of interacting class-diagrams are to be selected. That is, from a methodological point we are proposing a two-level based perception: first, each independent component is constructed and rapid-prototyped and then the interaction is dealt with by hiding all local features.
- Using the reflection capabilities of rewrite logic, we directly provide how message rewrite rules are to be performed, where carefully chosen strategies will correspond to life-cycle diagrams. To capture sequence diagrams, we have to add to the list of attributes in each class-diagram a particular attribute we called state and construct an appropriate strategy reflecting its change.

The rest of this paper is organized as follows. Using a very simplified example, in the next section we present an overview of different UML diagrams we will be focusing on. In the third section, we concentrate on the syntactical integration of OCL descriptions into class-diagrams. In the four section we propose an adequate interpretation of this integration in terms of the extended MAUDE language. The last section recapitulates the achieved work and discusses some future improvements. Unfortunately, due to space limitation we could not presents the semantical part, that is the rewrite theory, of the MAUDE extension and the meta-level for capturing state-chart diagrams semantics; however, the extended version of this paper adressing these two issues is appearing as a technical report [AS02].

2 UML Diagrams through a Simplified Example

In this section we present a simplified illustration of different UML diagrams we are concerning with, namely class-, object-, and associated OCL descriptions. From a methodological point of view, the construction of such diagrams has to be seen as a first phase towards the modelling / validation of any system. In this simplified banking system we assume having two ‘independent’ (i.e. related only through relationships) class-diagrams, namely the account and the account owners diagrams. Before giving the detail of each diagram, the left-hand side of Figure 1 sketches the ‘generic’ form of class diagrams—where classes are composed as usual of a set of attributes and operations and may be related to each other through inheritance, role and associations.

Using this general form, the right-hand side of Figure 1 depicts the class-diagram of an account class hierarchy. In this hierarchy we have as a super-class `current accounts`. Attributes of this class are the `balance`, the account `owner`’s identity and a constant, denoted by `Limit`, as a minimal value of the balance. As methods of this class we consider : the opening and deletion of any

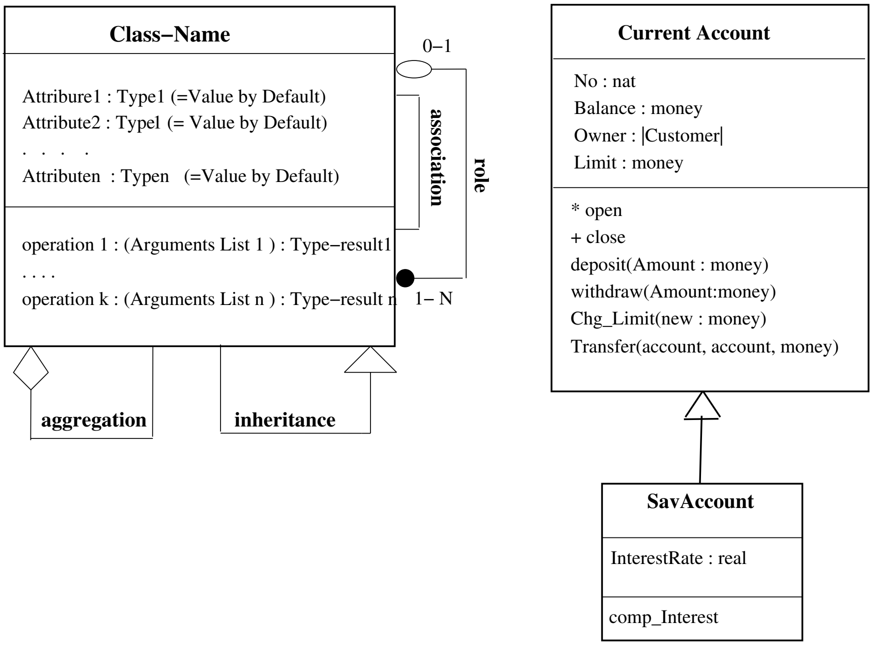

Fig. 1. The Generic form of UML class diagrams with the accounts example

account, the deposit of a given amount, the withdraw of a given amount, and the transfer of funds from an account to another. As a subclass we consider the class `Sav-Account` which is characterized by the interest percent. The interest percent of the balance is added up (at the end of each year for instance) to the current balance through the method `comp_interest`.

As a sketch of the OCL description part which will be the interest of our focus, we present in what follows the corresponding description to be associated, for instance, with the `transfer` method. This description is depicted in Table 1, where besides the signature of the method and its informal meaning, relevant is the condition `Pre` to be true to perform such a transfer, namely the account source balance has to the greater than the intended amount to be transfered. Relevant is also, the result of any operation, denoted by `Post`.

3 Integration of OCL Descriptions into Class-Diagrams

As we pointed out above, modelling separately OCL descriptions, and specifically different details about methods, does not only prevent a full respect of the object-oriented philosophy—that is, an *intrinsic* description of structural and behavioural aspects— but also prohibits any form of validation by rapid-prototyping. Indeed, it is very desirable that such a validation is performed at the specification level without requiring further refinements or implementations.

Table 1. A simplified illustration of OCL description using the transfer method

keywords	**corresponding instantiation**
`Operation`	Account :: transfer (src:Account, dest:Account, amount : Money)
`Description`	The system takes amount from the source, if there is, and places it on the destination
`Pre:`	$\text{src.balance} \geq \text{amount}$
`Post:`	$\text{src.balance}- = \text{amount} \wedge \text{dest.balance}+ = \text{amount}$

However, we should be aware that although several OO existing modelling frameworks do achieve such intrinsic integration, only a few of them offer an appealing and high-comprehension level provided by UML diagrams. In other words, our objective is to *maintain* all the strengths of UML diagrams and just *enriching* them in such a way that OCL descriptions concerning operations could be smoothly *merged* in the class-diagrams. In the following we present step by step this enrichment of class-diagrams with related OCL descriptions.

3.1 Enrichment of Class-Diagrams by Variables and Scopes

The first step towards integrating behavioural aspects in UML class-diagrams consists in the following. In order to allow controlling the change of attribute values as well as the invoked objects and values of message parameters, we propose to endow each attribute (resp. operation parameter) with at least one variable which has to be of the same sort. Besides argument variables, we also make explicit the objects (identities) invoked in a given message. On the other hand, as we mentioned we want rather a component-oriented perception. To this aim, we associate with each attribute (resp. operation) a scope which may be *local* or *observed*—shortly `l` or `o`. Finally, in order to distinguish between invoked objects in a given operation (as in the transfer operation for instance), we also propose to include in the attribute box a list of (current) identifier variables preceeded by the (key)word `Identity`.

These enrichment are depicted in Figure 2, where with respect to the generic general form of class-diagrams we already introduced in Figure 1 we have added variables and scopes with each attribute and operations. In this enriched general form we have also separated (by using two boxes) between messages considered as local and those considered as observed ones.

Example 1. By restricting the account class-diagram to just the current accounts class, in the left-hand side of Figure 3 we have enriched this class by different variables for attributes as well as for message arguments. Also, we have distinguished between local and observed attributes and messages. However, the user may always change the scope of such attributes and messages at a need; for instance we have decided that the `transfer` operation be an observed one just for illustration purpose as will be subsequently made clear. In the right hand side, we have introduced a new 'enriched' class-diagram, namely the account owners' (or customer) class.

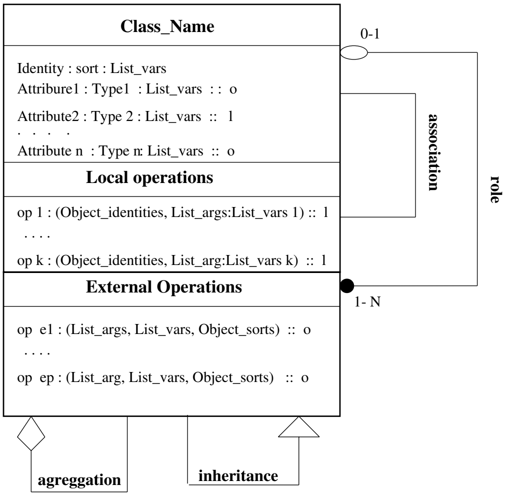

Fig. 2. The generic UML class-diagrams enriched by variables and scopes

3.2 Introducing New Notations for Behavioural Aspects

After enriching class-diagrams with the notions of variables and scopes, the next step consists in intrinsically incorporating in these diagrams the dynamics of each operation instead of (or after) describing them separately using OCL descriptions. In the endeavor to achieve this crucial step, we propose to add new semi-graphical notations we borrow from Petri-nets ones [Rei85]. More precisely, with respect to our objective of enhancing scalability and component-orientation, first, we present how class-diagrams' behaviour is conceptualized, and then we deal with the behaviour gouverning the interaction between 'independent' class-diagrams composing the whole system.

Internal behaviour within class-diagrams. As described in Figure 4, the incorporation of the dynamics associated with each local message—all observed messages are ignored at this level—consists in constructing a Petri-net like transition, with the following characteristics.

- The transition form we associate with each operation is represented as a rounded box. Within each rounded box we associate a condition (i.e. a boolean expression), we denote by `Mes_cond`, which has to be built on the invoked attributes and message argument variables using different comparison operators (e.g $=, >, <, \neq, \leq, \geq$) and / or boolean operators (e.g `and`, `or`).
- The (input) arrows or arcs going from the class to each rounded box are labelled by two information. On the one hand, the first inscription denoted by `Invoked_Mes` is to be always a local message of the form $\texttt{op}_i$`(Id1, .., var1,..,vark)`; where $\texttt{op}_i$ is any operation or message declared as local one in the corresponding class-diagram, and the parameters `Id1, ..,`

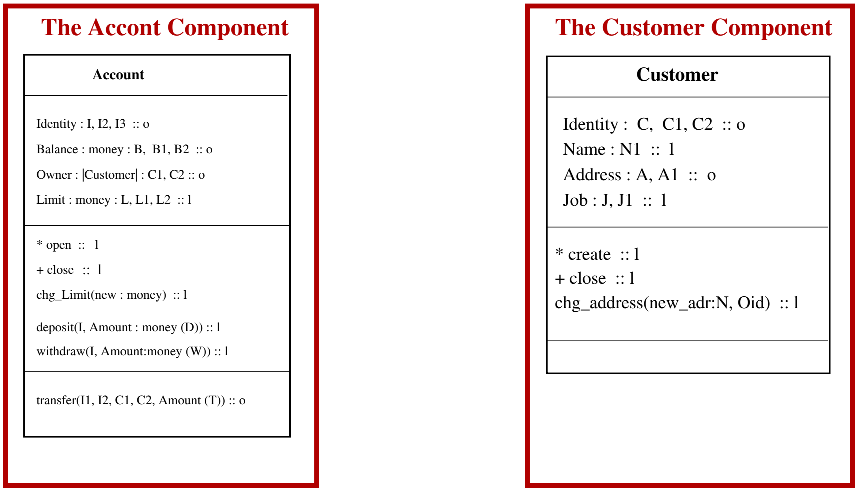

Fig. 3. The accounts and customers class-diagrams with variables and scopes

`var1,..,vark` have to reflect the object identifiers and other invoked parameters. The second inscription we denote by `Invoked attributes` has to be of the form:

$$\texttt{Id}_1\texttt{.atr}_1\texttt{:Var}_1\ ,\ldots,\ \texttt{Id}_k\texttt{.atr}_k\texttt{:Var}_k$$

Intuitively each pair $\texttt{Id}_i\texttt{.atr}_i\texttt{:Var}_i$ corresponds to an invoked attribute belonging to an object $\texttt{Id}_i$ with $\texttt{Var}_i$ to be understood as a current value. The selected pairs should correspond to objects (identifiers) invoked in the corresponding message. In other words, they have to be involved either for changing these current values or participating in the condition. This will play an important role towards exhibiting intra-object concurrency as we show later.

- Finally, the inscription associated with the output arrow, we denoted by `Result_change`, has to be of the form.

$$\texttt{Id}_1\texttt{.atr}_1\texttt{:Exp}_1\ ,\ldots,\ \texttt{Id}_k\texttt{.atr}_k\texttt{:Exp}_k.$$

  Each expression $\texttt{Exp}_i$ has to reflect the intended change of the corresponding value of the invoked attributes.

Example 2. Following this general form in integrating message dynamics into class-diagrams, Figure 5 illustrates the incorporation of different behaviour associated with local operations in both `Account` and `Account Owners` classes. For instance, to reflect the withdraw behaviour, first, we have to select an account and a corresponding amount: this fact is illustrated by the inscription `withdraw(I,M)`, with `I` as account identifier and `M` the associated amount to

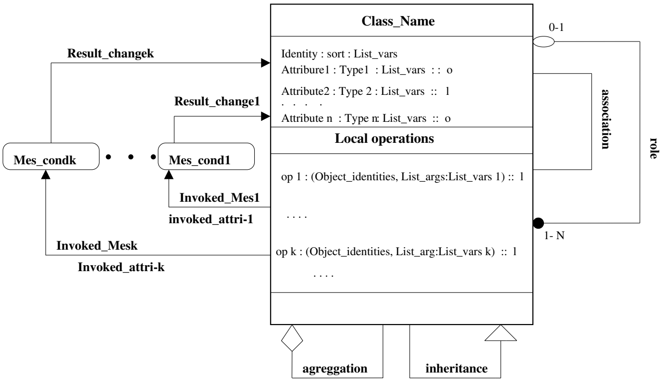

Fig. 4. The general form of enriching class-diagram by operations dynamic

be withdrawn. The second inscription, labelling the corresponding input arc of this method, namely `I.balance:B, I.limit:L`, involves the attributes of the invoked object (i.e. `I`) which are needed to express the corresponding state's change and conditions. As a condition of this method we require that the current value of the balance should by greater that `M` and the difference `B-M` be greater than `L`. Finally the resulting state's change has to be `I.balance:B-M, I.limit:L` which corresponds to the output arc inscription.
On the light of this explanation, all other operation dynamics are constructed following the same reasoning. It is worth-noting that all observed messages are simply omitted at this level.

Interaction between independent class-diagrams. As we pointed out in the introduction, we are proposing a two-level based methodology for integrating different UML diagrams. That is, after enriching each independent class-diagram with the appropriate behaviour as a first level, the next step is to deal with the interaction between different class-diagrams composing the whole system. To this purpose, we introduce very similar constructions with the following specificities. First as depicted in Figure 6, at this level in each class-diagram we have to deal only with those attributes and operations chosen to be observed. That is, the already constructed internal behaviour as well as all local attributes and messages have to be hidden at this inter-class diagrams' interaction level. Second, besides observed messages also relationships relating different class-diagrams may have corresponding behaviours. Third, technically the construction of such behaviour is exactly as for local messages except that now more than one class-digram is needed.

Example 3. In Figure 7 we have constructed the corresponding behaviour of the *transfer* message. In this construction we require for instance that for performing any money transfer between two accounts their corresponding owners should

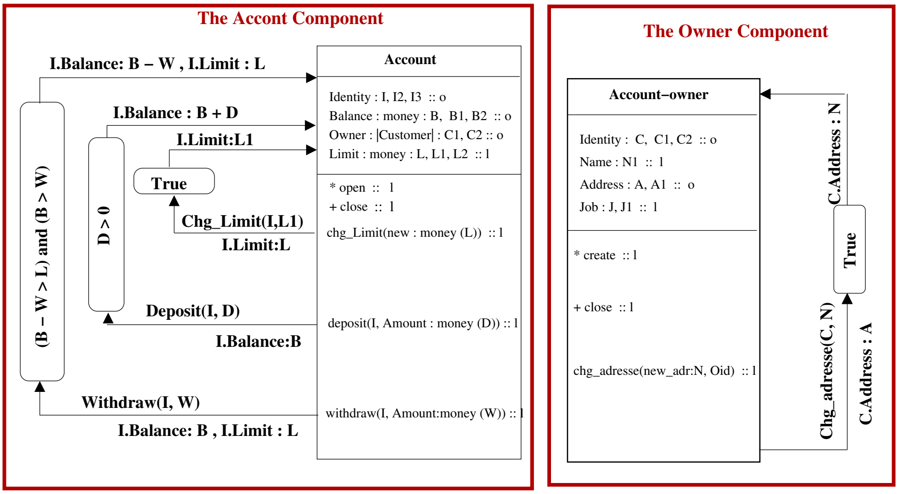

Fig. 5. The Account and Owner class-diagrams extended by messages behaviour

have the *same* address (the same city). With such a constraint we should now also involve the owner class-diagram.

4 Interpretation of the Proposed Integration in the Extended Maude

First as we pointed out in the introduction, due to space limitation we assume the reader fimiliar we the Maude language and the extension for intra-object concurrency and componentization we proposed in [AS99a]. This section is devoted to the theoretical underpinning of the proposed syntactical modelling artifacts. Our objective is to propose a semantical framework that allows fulfilling all the mentioned features, namely : (1) an indivisible integration of structural and behavioural aspects of objects within classes ; (2) a full exhibition of intra- and inter-object concurrency; (3) a satisfactory interpretation of all structuring abstractions within the enriched class-diagrams; (4) a clean separation between the internal description and reasoning within any class-diagram and the description and reasoning about the interaction between such class-diagrams. By reasoning we mainly understand the rapid-prototyping using the deduction rules of such an adequate semantical framework.

The semantical framework we are proposing is based on rewrite logic [Mes92], which has been proved very appropriate for dealing with concurrent OO systems in the recent years [Mes98]. Another advantage that makes this logic very practical is the current implementation of the Maude language [CDE+99], those programs are just theories in this logic.

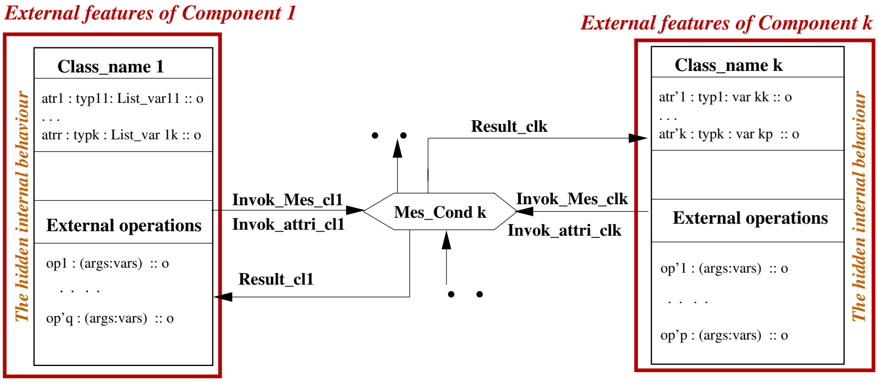

Fig. 6. The Interaction between independent class-diagrams

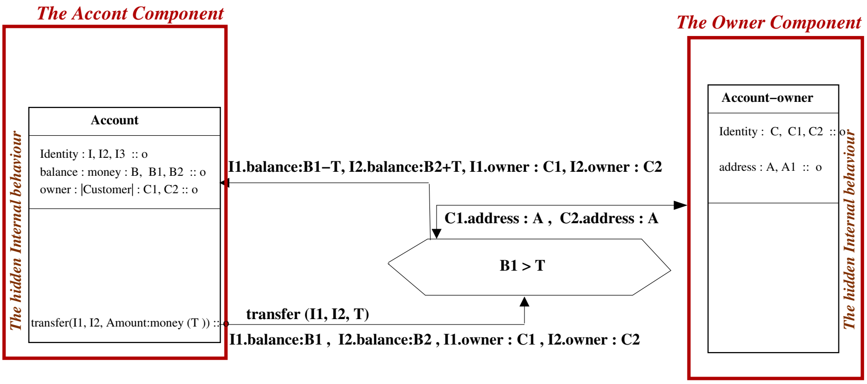

Fig. 7. The interaction between the account and the customer class-diagrams

More precisely, in the next subsection we will focus more on the way of translating the proposed integration to the extension of Maude we proposed in [AS99a, AS99b], which allows fulfilling all the mentioned (four) objectives.

4.1 Translating Extended Class-Diagrams into Maude

This subsection is devoted to the translation of our proposed variant of class-diagrams into the Maude language. To this aim, following the two-level suggested methodology, first, we have to deal with the translation of each independent class-diagram separately. Then, we should complete this translation by coping with the interaction between such independent class-diagrams composing the whole system.

Translating class-diagrams. By examining the general form of class-diagrams we proposed in Figure 4 and the Maude description it follows that such a translation is very straightforward. More precisely, the steps to be performed are the following.

1. The translation of structural aspects of any class-diagram is directly captured as a MAUDE module, where attributes are to be declared with their corresponding sorts and operations are conceived as messages. Besides that, in order to distinguish local attributes / messages and observed ones, instead of the OO MAUDE modules' keywords *class* and `Msgs` we will rather use `class-loc` and `class-obs` as well as `Msg-loc` and `Msg-obs`.
2. The translation of the behaviour we associated with each operation can also be intuitively expressed as a rewrite rule. More precisely, we have to perform the two follwoing two steps:
   - first, we have to reorganize input (resp. ouput) inscriptions from the form
     $$\mathtt{Id}_1.\mathtt{atri}_1 \ \mathtt{:}\ \mathtt{val}_1,\ \ldots,\ \mathtt{Id}_k.\mathtt{atri}_k \ \mathtt{:}\ \mathtt{val}_k$$
     (resp. $\mathtt{Id}_1.\mathtt{atri}_1 \ \mathtt{:}\ \mathtt{exp}_1,\ \ldots\ ,\ \mathtt{Id}_l.\mathtt{atri}_l\mathtt{:}\mathtt{exp}_l$) to the MAUDE form one, namely:
     $$\langle Id_1|atri_1 : val_1\rangle, \ldots, \langle Id_k|atri_k : val_k\rangle$$
     (resp. $\langle Id_1|atri_1 : exp_1\rangle, \ldots, \langle Id_l|atri_l : exp_l\rangle$);
   - each Petri-net like transition is then to be expressed as a rewrite rule of the form:
     $$\mathtt{Invoked\_Mes}_i \quad \mathtt{Invoked\_Attri}_i \quad \Rightarrow \mathtt{Result\_change}_i \quad \mathtt{if} \quad \mathtt{Condition}$$

Example 4. With respect to these straightforward ideas, the corresponding structural MAUDE part of the account class, for instance, takes the following form:

```
omod Account is
  protecting Money .
  class-loc Account | Limit : Money .
  class-obs Account | Balance : Money, Owner : OId .
  msg-loc  Chg_Limit : OId  Money → Msg .
  msg-loc  Deposit : OId  Money → Msg .
  msg-loc  Withdraw : OId  Money → Msg .

  vars I, I2, I3 : OId .
  vars C, C1, C2 : OId .
  Vars  B, B1, B2, L, L1, L2 : Money .
  vars W, D : Money .
```

On the other hand, following these very simple translating ideas, the corresponding rewrite rules of the messages in the account class-diagram, for instance, are as follows. In these rules we have considered the corresponding message names as rule labels.

`Chg_Limit` : $Chg\_Limit(I, L1)\ \langle I|Limit : L\rangle \Rightarrow \langle I|Limit : L1\rangle$
`Deposit` : $Deposit(I, D)\ \langle I|Balance : B\rangle \Rightarrow \langle I|Balance : B + D\rangle$ `if` $(D > 0)$
`Withdraw` : $Withdraw(I, W)\ \langle I|Balance : B, Limit : L\rangle \Rightarrow \langle I|Balance : B - W, Limit : L\rangle$ `if` $(B - W > L) \wedge (B > W)$

Translating inter-class interactions. The translation into extended MAUDE of the inter-class interactions is like the translation of intra-class structure and behaviour except that here we deal only with observed features in each class. Besides that, in order to separate the invoked messages and objects in each class, we adopt the notation

$$(\texttt{Class\_name}_1,\ \texttt{configuration}_1) \otimes \dots \otimes (\texttt{Class\_name}_k,\ \texttt{configuration}_k)$$

Following that, the general form of rewrite rules to associate with the interaction pattern depicted in Figure 6 takes the following configuration:

$$(\texttt{Class\_name}_1,\ \texttt{Invok\_Mes\_cl1}\ \ \texttt{Invok\_attri\_cl1}) \otimes \dots \otimes (\texttt{Class\_name}_k,\ \texttt{Invok\_Mes\_clk}\ \ \texttt{Invok\_attri\_clk}) \Rightarrow (\texttt{Class\_name}_1,\ \texttt{Result\_cl1}) \otimes \dots \otimes (\texttt{Class\_name}_k,\ \texttt{Result\_clk})\ \ \texttt{if}\ \ \texttt{Mes\_Condk}$$

Example 5. Using the above general of rewrite rule, the rule corresponding to the observed message `transfer` in Figure 7 takes the form:

`Transfer:` $(Account, transfer(I1, I2, T)\langle I1|balance : B1, Owner : C1\rangle\langle I2|balance : B2, Owner : C2\rangle) \otimes (Accout - owner, \langle C1|address : A\rangle\langle C2|address : A\rangle) \Rightarrow (Account, \langle I1|balance : B1 - T, Owner : C1\rangle\langle I2|balance : B2_T, Owner : C2\rangle) \otimes (Accout - owner, \langle C1|address : A\rangle\langle C2|address : A\rangle)$ `if` $(B1 > T)$

5 Conclusions

In this paper, we have proposed a sound and intuitive integration of all relevant UML-diagrams for dealing complex distributed information systems. More specifically in our integration we have concentrated on object- and class-diagrams and OCL descriptions in particular pre- and post-conditions. Beside being syntactically and semantically well-founded, the proposed integration enhances concurrency with a full exhibition of intra- as well as inter-object concurrency, componentization as we explicitly separate between internal and object features in any enriched class-diagram, and rapid-prototyping of this coherent view of different system diagrams using rewriting techniques.

Methodologically, this proposed sound integration has to be regarded more as an intermediate phase between the UML modelling and implementation phases. That is, after specifying any complex information systems using UML diagrams, a semi-automatic translation or integration of these diagrams following the explained steps allows achieving at least the three above mentioned objectives. We argue that fulfilling such goals promotes more reliability, reusability and eliminate different errors and misunderstanding at an early stage.

As a future perspectives, first we are conscious that this proposal is just a first stone in bringing more coherence and reliability to the UML methodology, and it has to be improved, extended and be supported by appropriate software tools. In this sense, firstly we are currently working on more complex non-trivial studies to assess and enhance the practicability of this proposal. Such case studies have also to be validated using the current implementation of the MAUDE language. As a very promising extension we are working on dealing with dynamic evolution of such integration using the rewriting logic meta-level. This will offer in particular to change in a runtime way the scope, the internal behaviour as well as the interaction between different components.

References

[ACR00] E. Astesiano, M. Cerioli, and G. Reggio. Plugging data constructs into paradigm-specific languages: Towards an application to UML. In T. Rus, editor, *Proceedings Algebraic Methodology and Software Technology, 8th International Conference, AMAST 2000, Iowa City, Iowa, USA, May 2000,* volume 1816 of *LNCS,* pages 273-292. Springer, 2000. 297

[Aou00] N. Aoumeur. Specifying Distributed and Dynamically Evolving Information Systems Using an Extended Co-NETs Approach. In G. Saake, K. Schwarz, and C Tärker, editors, *Transactions and Database Dynamics,* volume 1773 of *Lecture Notes in Computer Science, Berlin,* pages 91-111. Springer-Verlag, 2000. *Selected papers from the 8th International Workshop an Foundations of Models and Languages for Data and Objects, Sep. 1999, Germany.* 297

[AS99a] N. Aoumeur and G. Saake. On the Specification and Validation of Cooperative Information Systems Using an Extended MAUDE. In K. Futatsugi, J. Goguen, and J. Meseguer, editors, *Proc. of 1 st Int. OBJ/CafeOBJ/Maude Workshop, at FM'99 Conference, Toulouse, France,* pages 197-211. The Theta Foundation Bucharest, Romania, 1999. 298, 305, 306

[AS99b] N. Aoumeur and G. Saake. Operational Interpretation of the Requirements Specification Language ALBERT Using Timed Rewriting Logic. In *Proc. of 5th Int. Workshop an Requirements Engineering: Foundation for Software Quality (REFSQ'99), Heidelberg, Germany.* Presses Universitaires de Namur, 1999. 306

[AS99c] N. Aoumeur and G. Saake. Towards an Object Petri Nets Model for Specifying and Validating Distributed Information Systems. In M. Jarke and A. Oberweis, editors, *Proc. of the llth Int. Conf. an Advanced Information Systems Engineering, CAiSE'99,* volume 1626 of *Lecture Notes in Computer Science,* pages 381-395. Springer-Verlag, 1999. 297

[AS02] N. Aoumeur and G. Saake. Integrating and Rapid-prototyping UML Structural and Behavioural Diagrams Using Rewriting Logic. Preprint, Fakultät für Informatik, Universität Magdeburg, March 2002. 299

[Ast99] E. Astesiano. Algebraic Specification of Concurrent Systems. In E. Astesiano, 11.4. Kreowski, and B. Krieg-Brückner, editors, IFIP *14.3 Volume an Foundations of System Specification, Chapter 1. To appear in Springer LNCS,* 1999. 297

[BJR98] G. Booch, I. Jacobson, and J. Rumbaugh, editors. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1997. 296

[BJR97] G. Booch, I. Jacobson, and J. Rumbaugh, editors. *Unified Modeling Language, Notation Guide, Version* 1.0. Addison-Wesley, 1998. 296

[CDE+99] M. Clavel, F. Duran, S. Eker, J. Meseguer, and M. Stehr. Maude. Specification and Programming in Rewriting Logic. Technical report, SRI, Computer Science Laboratory, March 1999. URL: http://maude.esl.sri.com. 305

[CRSS98] S. Conrad, J. Ramos, G. Saake, and C. Sernadas. Evolving Logical Specification in Information Systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems,* chapter 7, pages 199-228. Kluwer Academic Publishers, Boston, 1998. 297

[EK00] A. Evans and S. Kent. Proc. *3rd Int. Conf. Unified Modeling Language (UML'2000).* LNCS. Springer, 2000. 297

[FR99] R. France and B. Rumpe. Modeling dynamic software components with UML. In *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings,* volume 1723 of *LNCS.* Springer, 1999. 297

[JSHS96] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. TROLL - A Language for Object-Oriented Specification of Information Systems. *ACM Transactions an Information Systems,* 14(2):175-211, April 1996. 297

[Mes92] J. Meseguer. Conditional rewriting logic as a unified model for concurrency. *Theoretical Computer Science,* 96:73-155, 1992. 305

[Mes98] J. Meseguer. Research Directions in Rewriting Logic. In U. Berger and H. Schwichtenberg, editors, *Computational Logic, NATO Advanced Study Institute, Marktoberdorf, Germany.* Springer-Verlag, 1998. 305

[Mos97] P. Mosses. Cofi: The common framework initiative for algebraic Specification and development. In Proc. Intl. Symp. an *Theory and Practice of Software Development (TAPSOFT),* volume 1214 of *LNCS,* pages 115-137. Springer, 1997. `COFi Homepage: http://www.brics.dk/Projects/CoFI`. 297

[RCA01] G. Reggio, M. Cerioli, and E. Astesiano. Towards a rigorous semantics of UML supporting its multiview approach. In H. Hussmann, editor, *Fundamental Approaches to Software Engineering. 4th International Conference, FASE 2001 Held as Part of the Joint European Conferences an Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6. 2001 Proceedings,* volume 2029 of *LNCS,* pages 171-186. Springer, 2001. 297

[Rei85] W. Reisig. Petri Nets. *EATCS Monographs an Theoretical Computer Science,* 4, 1985. 302

[WMB99] A. Wienberg, F. Matthes, and M. Boger. Modeling dynaanic software components with uml. In Robert France and Bernhard Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings,* volume 1723 of *LNCS,* pages 204-219. Springer, 1999. 298