IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4265

Dimensions in the Object Oriented Software Development Process

Claudia Pons, Roxana Giandini, Gabriel Baum, Joe Luis Garbi and Paula Mercado
LIFIA, Laboratorio de Investigación y Formación en Informática Avanzada
Tel/Fax: 054 221 4228252, {cpons, giandini}@sol.info.unlp.edu.ar

ABSTRACT

During the object-oriented software development process, a variety of models of the system is built. All these models are not independent, but they are related to each other. Elements in one model have trace dependencies to other models; they are semantically overlapping and together represent the system as a whole.

In this paper we classify relationships between models along three different dimensions, proposing a formal description of them. The goal of the proposed formalization is to provide formal foundations for tools that perform intelligent analysis on models assisting software engineers through the development process. In this direction we discuss the construction of a tool, based on the formalization, supporting the verification of traces between requirement model and analysis models.

INTRODUCTION

A software development process, e.g. The Unified Process (Jacobson et al., 1999), is a set of activities needed to transform user's requirements into a software system. Modern software development processes are iterative and incremental, they repeat over a series of iterations making up the life cycle of a system. Each iteration takes place over time and it consists of one pass through the requirements, analysis, design, implementation and test activities, building a number of different artifacts (i.e models). All these artifacts are not independent; they are related to each other, they are semantically overlapping and together represent the system as a whole. Elements in one artifact have trace dependencies to other artifacts. On the other hand, due to the incremental nature of the process, each iteration results in an increment of artifacts built in previous iterations.

Different relationships existing between models can be organized along the following three dimensions:

- internal dimension (artifact-dimension).
- vertical dimension (activity -dimension)
- horizontal dimension (iteration-dimension)

The internal dimension deals with relations between sub-models that coexists consistently making up a more complex model. For instance, an analysis model consists of analysis class diagram, interaction diagrams, collaboration diagrams. All the artifacts within a single model are related and have to be compatible with each other.

The vertical dimension considers relations between models belonging to the same iteration in different activities (e.g. a design model realizing an analysis model). Two related models represent the same information, but from different abstraction level. Both related models also coexist and should be syntactically and semantically compatible with each other.

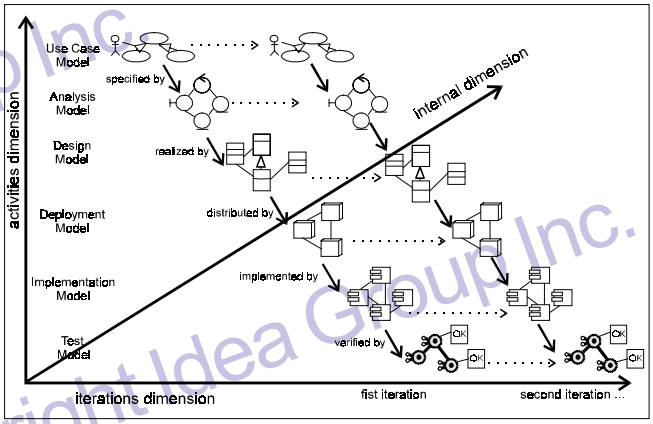
The horizontal dimension considers relations between artifacts belonging to the same activity in different iterations (e.g. a use case is extended by another use case). In this dimension new models are built or derived from previous models by adding new information that were not considered before or by modifying previous information.

Figure1 illustrates the three dimensions described above. It lists the classical activities (requirements, analysis, design, implementation and test) in the vertical axis and the sequence of iterations in the horizontal axis.

Relations between models should be formally defined since the lack of accuracy in their definition can lead to wrong model interpretations and inconsistency among models.

At the present the Unified Modeling Language UML is considered the standard modeling language for object oriented software development process. The specification of UML constructs and their relationships (UML, 2000) is semi-formal, i.e. certain parts of it are specified with well-defined languages while other parts are described informally in natural language. There is an important number of theoretical works giving a precise description of core concepts of UML and providing rules for analyzing their properties; see, for instance the works of Evans et al.(1998;1999), Kim and Carrington (1999), Breu et al.(1997), Knapp (1999), Övergaard (1999, 2000), Pons and Baum. (2000). These works improve precision of syntax and semantics of isolated UML models, without dealing with relationships between models.

Figure 1: Dimensions in the software development process



In addition, Övergaard and Palmkvist (1998, 2000), Petriu and Sun (2000) , Sendall. and Strohmeier, (2000) and Whittle et al. (2000) focus on relationships between different UML models. Following this direction, we classify relations between models along three different dimensions, proposing a formal description of them. This paper reports an extension of our earlier work described in Pons et al.(2000).

INTERNAL-DIMENSION RELATIONS

Every model is made up from a number of related sub-models (or artifacts) that have to be semantically compatible obeying to several constraints between them.

The UML specification document (UML, 2000) defines the abstract syntax of UML by class diagrams and well-formedness rules expressed in the Object Constraint Language OCL (UML, 2000). Most of the well-formedness rules in that document are examples of constraints on internal-dimension relations.

After a number of revisions, the UML specification document still contains ambiguities and inconsistencies. We have been analyzing

This paper appears in *Issues and Trends of Information Technology Management in Contemporary Organizations,* the proceedings of the Information Resources Management Association International Conference.

internal relationships between models in order to improve their specification, see for example (Cibran et al. 2000).

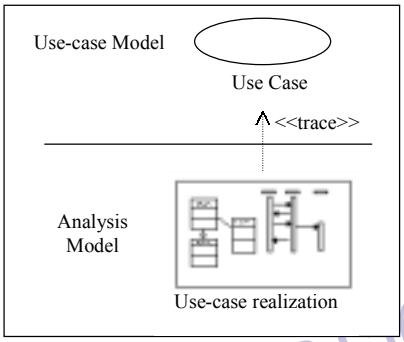
VERTICAL-DIMENSION RELATIONS

In this section we analyze vertical relations, that is to say relations between models belonging to the same iteration in different activities. Due to space limitations we only describe relationships between the requirement phase and the analysis phase.

Creating Analysis Models from Use Cases

A use case in the use-case model is realized by a collaboration within the analysis model that describes how a use case is realized and performed in terms of analysis classes and their interacting analysis objects. A use case realization has class diagrams that depict its participating analysis classes, and interaction diagrams that depict the realization of a particular flow or scenario of the use case in terms of analysis object interactions. Figure 2 shows the relation between a use case and its realization.

Figure 2: Use case realization



Example: We present the model of a system to maintain a Library. The members of the library share a collection of books. The system should allow them to borrow books, to return them or to renovate a loan. When returning or when renovating the loan of a book, the member should pay a fee. In the event this fee is not paid, the member won't be able to borrow a new book or to renovate a loan. Use case renewLoan specifies the functionality of the system, for the renew of a loan.

Use cases can be specified in a number of ways. Generally natural language structured as a conversation between user and system is used, see (Jacobson et al., 1993). The conversation shows the request of a user and the corresponding answers of the system, at a high level of abstraction. The following paragraph shows a conversation between an actor (a member of the library) and the system. The conversation considers the normal action sequence and also alternative sequences (e.g. the case in that the book is not available):

User Actions: User-asks-for-renew-loan

System Answers: alidate-member-id, validate-book-availability, ask-for-debt, renew-loan

Alternatives:. Member-identification-is-not-valid, then reject-loan. Book-is-not-available, then reject-loan. Member-has-debt, then ask-for-payment, then renew-loan.

In the UML a UseCase is a kind of Classifier having a collection of operations (with its corresponding methods). Operations describe messages that instances of the use case can receive. Methods describe the implementation of operations in terms of action sequences that are executed by the instances of the use case.

Let uc be the use case defined above. The definition of uc (using the standard notation and metamodel of UML (2000) is as follows:

uc.operations = <op1>

op1.name=ask for renew loan

op1.method.body= {<validate-member-d, validate-book-availability, ask-for-debt, renew-loan>,

< validate-member-id, reject -oan>, < validate-member-id, validate-book-availability, reject-loan>,

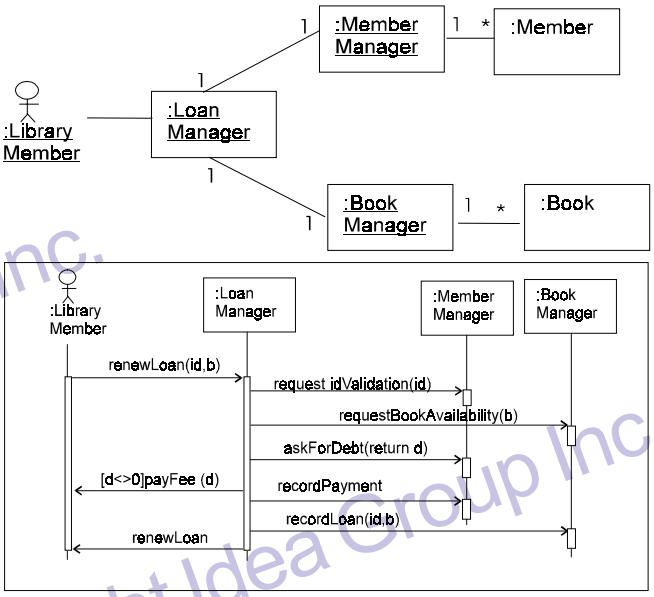
< validate-member-id, validate-book-availability, ask-for-debt, ask-for-payment, renew-loan >}

In general we abbreviate op.method.body by op.actionSequence. The body of a method is a procedure expression specifying a possible implementation of an operation. The definition of procedure expressions is out of the scope of UML, here we interpret a procedure expression as a set of action sequences.

The Realization of the Use Case:

Figure 3 shows a Collaboration model including a set of Classifier Roles and their connections, and one of the iteration diagrams specifying the message flows between objects playing the roles in the collaboration. These diagrams are expected to realize the use case above; this fact will be formally proved in next section.

Figure 3: A collaboration realizing the use case



Formalizing the realization relation between Use Cases and Collaborations

Lets define a set of concepts that are necessary in order to formalize the relations between use cases and collaborations.

Def. 1: let (MS,^{3}) be the poset of messages in an interaction (messages are partially ordered by the predecessor/successor relation). The set of linearizations on MS is defined as the set of sequences of messages in MS, and it is denoted as lin(MS,^{3}).

Def. 2: maxLin(MS,^{3}) is the set of maximal linearizations on MS. It is obtained from lin(MS,^{3}) by dropping every sequence that is contained in another sequence in the set.

Def. 3 : let S be a set of sequences of actions. external(S) denotes the sequences of S obtained omitting all the actions that are not visible externally.

Def. 4: a conformance declaration is a correspondence between action names in a use case and action names in a collaboration. Each name in the use case is mapped to (a name of) an action in the collaboration. This mapping provides more flexibility in the development process allowing analysts to modify the name of the actions as the process evolves.

For example, the following is a conformance declaration between the Use Case and the Collaboration above:

δ : Actions in the UC→Actions in the collaboration
ask-for-renew-loan → renewLoan(id,b)
validate-member-id → requestIdValidation(id)
validate-book-av. → requestBookAvailability(b)
ask-for-debt → askForDebt(ret d)
ask-for-payment→ payFee(d)
renew-loan → renewLoan
reject-loan → reject

At this point we can define the realization relation between a Collaboration C and a Use Case UC. A Use Case is realized by a Collaboration if the Classifiers Roles in the Collaboration jointly cooperate to perform the behavior specified by the Use Case, but not more. In the case that the Collaboration includes more behavior than the one specified by the Use Case, the Use Case would be only a partial specification of the behavior described by the Collaboration. On the other hand, a use cases specifies actions that are visible from outside the system, but do not specify internal actions, such as creation and destruction of instances, communication between internal instances, etc. (for example, recordPayment and recordLoan are internal actions)

Definition 5: A collaboration C is a realization of a Use Case UC according to the conformance declaration δ , denoted $C \geq_{\delta} UC$, if both of the following hold:

a- $\forall uo \in UC.operation. \forall ut \in uo.actionSequence. \exists int \in C.interaction. \exists ms \in lin(int.message).$ $(\delta(uo.name)=act.operation.name \wedge \delta^{+}(ut)=external(ms.tail.actions))$

b- $\forall int(C.interaction. \forall ms(maxLin(int.message). \exists uo \in UC.operation. \exists ut \in uo.actionSequence.$
$(\delta(uo.name) = act.operation.name \wedge \delta^{+}(ut)=external(ms.tail.actions)$

Where:
act = (ms.head).action,
ms.head is the first element in the sequence ms,
ms.tail is the subsequence obtained from ms by dropping the first element,
ms.actions is an abbreviation for ms.collect (e.action)
$\delta^{+}(ut)$= ut.collect (δ (a))

Definition above states that every action sequence specified by the Use Case must have a corresponding action sequence in the Collaboration, that is equal to it (except for internal actions), and vice versa.

HORIZONTAL-DIMENSION RELATIONS

In this section we analyze horizontal relations, that is to say relations between models belonging to the same activity in different iterations.

Evolving the Use-Case Model

A use case model may be evolved in different ways. The UML considers at least two forms of evolution: the extends and the generalization relationships between use cases. In this paper we only take into consideration the former.

The extend relation represents the enrichment of a use case by the definition of additional actions. An extend relationship from use case A to use case B indicates that an instance of use case B may include (constrained by specific conditions specified in the extension) the behavior specified by A.

The definition of extend includes both a condition for the extension and a reference to an extension point in the target use case, that is, a position in the use case where additions may be made. Once an instance of a (target) use case reaches an extension point to which an extend relationship is referring, the condition of the relationship is evaluated. If the condition is fulfilled, the sequence obeyed by the use-case instance is extended to include the sequence of the extending use case.

Evolving the Collaboration Model

The UML does not consider special dependency relationships between Collaboration. However since Collaborations realize Use Cases, it is important to reflect the relationships between Use Cases (e.g. extend relationships) on its realizing Collaborations. As well as Use Cases are extended by adding actions (defined in other Use Case), Collaborations can be extended with additional message sequences specified in another Collaboration.

For further details about the extension relationship between Collaborations based on the corresponding extension relationship between Use Cases, readers are referred to (Giandini et al., 2000).

TOOL SUPPORT: THE DEPENDENCY RELATIONS CHECKER

Tracing elements between different models is not an easy task. An essential element to perform this activity is support offered by CASE tools. In particular, it is useful to deal with formal mechanisms that contribute to a precise and rigorous verification of the relationship.

We built a tool named DRC (Dependency Relations Checker). DRC verifies the relation between the elements in the use case model and the analysis model. The DRC is based on the formal foundation for dependency relationships.

The main idea of the Dependency Relations Checker is to provide a friendly approach to formal verification of the dependency relationships that exist between different UML models.

In particular, the current version of DRC is focused on verifying if a given use case realization actually corresponds to a specific use case diagram. In other words, DRC verifies if a use case realization represents the behavior established by a use case diagram.

DRC takes place in the early phases of the development process, between the requirements phase and the analysis phase. It helps developers to verify consistency among the two phases, with the benefit of the formal background.

The Main Components of the Dependency Relations Checker

DRC does not provide a complete environment for the specification of models. This functionality is provided by the support of CASE tools, like Rational Rose. Our tool can be extended to allow the construction of UML models. However, the current approach provides a flexible working mechanism and allows an easy insertion of our tool in the development process, since it deals with the Rational Rose, a highly accepted tool in industry.

Clearly, we have two distinguished layers: the specification layer given by Rational Rose, that represents the DRC input and the verification layer, that performs checking activities that generate the final result. In the next section, we concentrate on the verification layer.

This section describes the main components of the verification layer. DRC was thought as a set of entities that intercollaborate to perform the global task. Figure 4 shows this relationship.

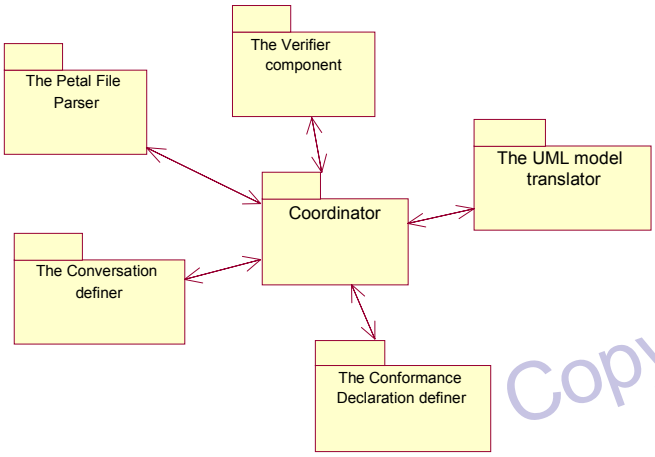
The components are, namely: the Petal File parser, the UML model translator, the Conversation definer, the Conformance Declaration definer and the Verifier component. All these modules are coordinated by a "coordinator" component and they all were implemented in the Smalltalk environment VisualWorks version 3.0. The role of each component of the DRC is described below.

The Petal File Parser. The Dependency Relations Checker takes the UML models defined with Rational Rose/C++ Version 4.0.3. In order to create an internal representation of the UML model that the Verifier component understands, the Petal File Parser component parses the content of the files generated by the Rational Rose (MDL files) and generates a collection of objects that represents it.

The UML model translator. Once the Petal File Parser has translated the file to an object collection, it is taken by the UML model translator component. This component translates the object representation of the Petal file to a simplified instance of the UML metamodel that represents the models generated in the requirement phase and in the analysis phase.

This component also has the responsibility of noticing any inconsistency among diagrams definitions, for example, the absence of

Figure 4: Main architecture of the tool



some diagram, its wrong construction, errors in the Conversation or Conformance Declaration syntax, among others.

The definition of these translation components gives a great flexibility to the design of the tool. It makes easier the task of extending the DRC to support different kinds of file formats provided by the great quantity of CASE tools used to specify models. In the other hand, it allows to remove and substitute these components by better ones, that perform the same task faster or more precisely.

The Conversation Definer. The definition of a Conversation is part of the construction of a Use Case Diagram. This task can be performed in both, the Rational Rose and the Dependency Relation Checker. In the Rational Rose, this is achieved attaching a "Note" to the Use Case Diagram, where you must manually write the Conversation using the given syntax. The DRC also provides a graphical interface that allows to create a Conversation for a given model.

The Conformance Declaration Definer. The Conformance Declaration establishes the correspondence between the actions in the use case and the actions in the collaboration.

The Verifier Component. The Verifier component takes as its input the UML model translated by the UML model translator. It also takes the Conversation and Conformance Declaration. The Verifier component performs the verification task of determining if a set of analysis diagrams are the realization of a use case diagram. This module applies the mathematical formulas mentioned above on the UML model and informs its conclusions.

It is important to say that the model can be easily extended to support different verifier components that perform other checkings using the same model. In this way, we can consider the tool of being an initial schema that allows different kinds of verification over UML models.

CONCLUDING REMARKS

Relations between software models should be formally defined since the lack of accuracy in their definition can lead to wrong model interpretations, inconsistency among models, inconsistent evolution of models, etc. In this paper we classify relations between models along three different dimensions (i.e. artifact dimension, activity dimension and iteration dimension), proposing a formal description of them.

The goal of the proposed formalization is to provide formal foundations for tools that perform intelligent operation on models. As an example of application of the formalization we have described the main components of the Dependency Relations Checker (DRC), that is a case tool giving support to verification of traces between use-case model and analysis model. This tool is very opened, due to the clear separation between its modules. This approach allows to easily extend or change its characteristics. This kind of tools represents an advance over the present state of the practice for project management.

REFERENCES

Breu,R., Hinkel,U., Hofmann,C., Klein,C., Paech,B., Rumpe,B. and Thurner,V., Towards a formalization of the unified modeling language. ECOOP'97 procs., Lecture Notes in Computer Science vol.1241, (1997).

Cibrán, M., Mola, V., Pons,C., Russo,W. Building a bridge between the syntax and semantics of UML Collaborations. In ECOOP´2000 Workshop on Defining Precise Semantics for UML France, June 2000.

Evans,A., France,R., Lano,K. and Rumpe,B., Towards a core metamodelling semantics of UML, Behavioral specifications of businesses and systems, Kluwer Academic Publishers, (1999).

Evans,A., France,R., Lano,K. and Rumpe, B., Developing the UML as a formal modeling notation, UML'98 Conference, Lecture Notes in Computer Science 1618, Springer-Verlag, (1998).

Giandini,R, Pons, C and Baum,G.. An algebra for Use Cases in the Unified Modeling Language. OOPSLA´00 Workshop on Behavioral Semantics, Minneapolis, USA, October 2000.

Jacobson, I., Christerson, M., Jonsson P. and Övergaard, G., Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, (1993).

Jacobson, I..Booch, G Rumbaugh, J., The Unified Software Development Process, Addison Wesley. (1999)

Kim, S. and Carrington,D., Formalizing the UML Class Diagrams using Object-Z, proceedings UML´99 Conference, Lecture Notes in Computer Sciencie 1723, (1999).

Knapp, Alexander, A formal semantics for UML interactions, Proceedings of the UML´99 conference, Colorado, USA,. Lecture Notes in Computer Science 1723, Springer. (1999).

Övergaard, G., and Palmkvist,K., A Formal Approach to Use Cases and Their Relationships. In UML'98 Conference, Lecture Notes in Computer Science 1618. Springer-Verlag, (1998).

Övergaard, G., A formal approach to collaborations in the UML. In UML´99 Conference, Colorado, USA,. Lecture Notes in Computer Science 1723, Springer. (1999).

Övergaard,G.. Using the Boom Framework for formal specification of the UML. in Proc. ECOOP Workshop on Defining Precise Semantics for UML, France, June 2000.

Overgaard G.and Palmkvist K. Interacting subsystems in UML, Proc. of The Third International Conference on the UML. LNCS. October 2000

Petriu,D and Sun,Y Consistent behaviour representation in activity and sequence diagrams. Proc. of The Third International Conference on the UML. LNCS. October 2000

Pons Claudia and Baum, Gabriel. Formal foundations of object-oriented modeling notations 3rd International Conference on Formal Engineering Methods, ICFEM 2000, IEEE Computer Society Press. Sept. 2000.

Pons , Claudia, Giandini, Roxana and Baum, Gabriel. Specifying Relationships between models through the software development process, 10th International Workshop on Software Specification and Design, USA, IEEE Computer Society Press. Nov. 2000.

Unified Modeling Language (UML) Specification - Version 1.3, March 2000. UML Specification, revised by the OMG, http://www.omg.org.

Sendall,S. and Strohmeier,A From Use cases to system operation specifications.. Proc. of The Third International Conf. on the UML, UK. LNCS. Oct 2000

Whittle, J.. Araújo, J.Toval, A Fernandez Alemán J.. Rigorously automating transformations of UML behavioral models, UML´00 Workshop on Semantics of Behavioral Models. UK, October 2000.