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
ITP4235

Towards A UML Profile For Building On Top of An Existing Application

Isabelle Mirbel
Laboratoire I3S, Route des Lucioles, France, Tel: 33 4 9294 2760, Fax: 33 4 9294 2896, mirbel@unice.fr

Violaine de Rivieres
Amadeus sas, France, Tel: 33 4 9294 7023, Fax: 33 4 9294 7171, vrebuffel@amadeus.net

INTRODUCTION

Applications are more and more often built on top of existing ones. Legacy applications and Enterprise Ressource Planning (ERP) are typical examples of development on top of a existing applications or components. It is very important to deal with the particularities of such context in the earliest stages of the process, and especially during the analysis phase, in order to control the risks inherent to this kind of development. Therefore, it is useful to clearly define as soon as possible what will be kept from the running application, why and how. Integration and interfacing aspects have also to be studied carefully.

Problems related to building on top of existing applications have been studied from the implementation point of view [Fo99,Be97,Op92]. But we believe that it has to be taken into consideration already during the analysis phase. Indeed, in addition to the code itself, the expertise about the functional domain and the interfaces (describing the relationships the existent application may have with others systems) may also be of interest, as it will be shown in this paper.

UML [OMG,RJB98,BRJ98] is an object-oriented graphical language. It is now a standard notation used by programmers as well as domain expert, through the whole development cycle. UML is customizable through the notion of profile [JBR98] which allows to re-assemble a set of extensions dedicated to a particular kind of application or development process. It has already been used in various domains [NF00,KS00]. In this paper, we present a profile dedicated to development on top of running applications.

The profile presented in this paper is part of the COOP methodology (Contextual Object-Oriented Process) where we propose a flexible approach for analysis and design with regards to the context of the application. The need for situation-specific approaches, to better satisfy particular situation requirements, has already been emphasized [SH96, RR01]. In COOP, the application context is described through different criterias; to develop the application on top of existing ones is one of these criterias. Flexibility is handled through the different fragments proposed in each phase of the process. Some of them are useful whatever is the context of the application, others are dedicated to the criteria. The profile presented in this paper is used through the fragments dedicated to applications developped on top of running ones. By situating the application in its context and by choosing the interesting fragments, the process is tailored for the application under consideration in order to allow a more efficient development process.

The paper is organized as follows. In section 2 we introduce a dedicated profile. Section 3 highlights the different aspects of preservation. Section 4 shows how this profile is used through the COOP methodology. Section 5 presents our conclusions.

A PROFILE FOR BUILDING ON TOP OF EXISTING APPLICATIONS

Our profile reassembles a set of stereotypes which has been defined to help in coping with problems related to developments on top of running applications: to maintain a clear distinction between existent parts and new developments, to present the application in an homogeneous way and to decide how the elements have to be preserved.

Use-Case Stereotypes

The main interest of use-case stereotypes is to clearly let understand what is new from what is preserved in the application under development. About elements taken from the existing applications, it is required to distinguish what is re-used as it is and what will be modified. It may also be useful to include in the models dealing with the analysis the description of elements which are not part of the application under development but may be useful to understand how the application will be working. It is important, in this case, to clearly indicate that these elements are included only for better understanding purpose. Therefore, the stereotypes dedicated to use-cases are the following ones.

- **New:** a use-case describing new functionalities.
- **To-be-modified:** a use-case describing functionalities already existing in the running applications, but enhanced through the new development under consideration.
- **Re-use:** a use-case describing existing functionalities reused as they are.
- **Out-of-scope:** a use-case describing functionalities not belonging to the application under study but useful to understand how the application works.

Constraints among stereotyped use-cases: if a use-case *uc1* describing the running applications is linked through an *inclusion* relationship to a use-case *uc2* stereotyped <<New>> or <<To-be-modified>>, then *uc1* must be stereotyped <<Re-use>> (and not <<Out-of-scope>>) because it is impacted by the new functionalities. On the contrary, if *uc1* is related through an *extension* relationship to another use-case *uc2*, then *uc1* must be stereotyped <<Out-of-scope>>, because it is not directly impacted by the new functionalities.

Actor Stereotypes

When dealing with the requirements of an application built on top of existing ones, it is required to specify the actors already interacting with the running applications (and still willing to continue the interaction) from the actors participating in the new functionalities. Most of all, systems already collaborating with the running parts of the application have to be identified, because they may already use an interface that must be kept compatible. These actors are a strong constraint for the new development. The proposed stereotypes help in distinguishing humans from systems and systems already interacting with the application from the other ones.

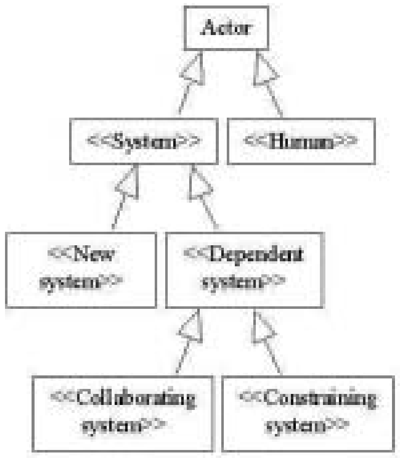
- **Human:** a person interacting with the application (through a man-machine interface).
- **System:** an actor interacting with the application and being a system (and not a person).
  - o **New system:** a system which will use the services of the application under development.
  - o **Dependent system:** a system already interacting with a running application and maintained through the new application. It is called a dependent system because it already exists and has still to be taken into account through the new development.
    - • **Constraining system:** a system already interacting with the running application. This stereotype indicates that the actor wants to continue the interaction exactly in the same

terms: the interfaces it uses must be kept compatible. The actor imposes constraints to the application.

• **Collaborating system:** a system already interacting with the application. The actor will continue to interact with the application but its interaction mode may be slightly modified.

The Figure 1 summarizes the different stereotypes dealing with actors, and their relationships.

Figure 1: Actor stereotypes



Class Stereotypes

When dealing with classes and associations, as well as with attributes and operations, it is necessary to distinguish what has to be developed from what is preserved from the running applications. As for the use-cases, we distinguish reused classes as they are from classes to be modified. We also complete the model by information about classes not belonging to the application under development, but useful to understand it.

- **New:** a new class, association, attribute or operation.
- **To-be-modified:** a class, association, attribute or operation to be modified. In the case of attribute, operation and association, a *note* has to be used to indicate what are the changes to be done on the element and what are the mapping with regards to the preservation.
- **Re-use:** a class, association, attribute or operation kept as it is.
- **Out-of-scope:** a class, association, attribute or operation described only for understanding purpose.

Package Stereotypes

Additional information given for understanding purpose may be useful to explain and justify the development to be done. Such information has to be isolated from the main part of the application, in a particular package/set of packages stereotyped <<Out-of-scope>>.

- **Out-of-scope:** a package dealing with information associated to the application only for understanding purpose.

Component Stereotypes

Stereotypes highlight components kept as is from the running applications as well as components enhanced and new components.

- **New:** a new component of the application.
- **To-be-modified:** a component taken from a running application and modified through the new development.
- **Re-use:** a component taken from a running application and kept as it is in the new application.

DEALING WITH RUNNING APPLICATIONS

Different aspects of running applications may be of interest during analysis. In addition to the *code* itself, the expertise about the *functional domain* (taken from the functionalities, data and screen shots of the application) may also be of interest. The *interfaces,* describing the relationships a running application may have with others systems (applications, databases, ...), may also be taken into consideration. By interfaces we do not include man-machine interfaces which are already included in the first aspect (dealing with the functional domain).

When dealing with legacy applications, for instance, the most important aspects are the expertise about the functional domain and the code, which has to be preserved and encapsulated. On the contrary, when an application is built with a Rapid Application Development tool, the code will not be preserved at all, but the expertise from the business, in addition to the man-machine interfaces, will be preserved. When starting a new development, the usual situation is the one which consists in enhancing an application, through a new version. In this case the three aspects (functional domain, interfaces, and code) have to be taken into consideration.

The analysis is dedicated to the clear identification of what has to be preserved, while the design indicates how such a preservation can be handled. To carry on this work, it is important to determine if the running part of the application:

- has to be reused as it is; for instance, while interfacing an existing component without any possibility to modify it (bought component);
- can be slightly modified; for instance, while interfacing an existing component, developed by the company, but used by other applications;
- can be widely modified; for instance, when an application evolves into a new version (and the development team is the owner of the existing part).

The three aspects of preservation (functionalities, interfaces, code) may be modulated in order to be better exploited during the development. We qualify each of them by: (i) *strong* when no modification is allowed, (ii) *medium* when modifications are allowed inside given boundaries, and (iii) *weak* when modifications are allowed.

HANDLING THE PRESERVATION THROUGH THE COOP METHODOLOGY

The COOP methodology is a flexible approach for analysis and design with regards to the context of the application. The context is described through different criterias; to develop the application on top of existing ones is one of these criterias. To develop successfully an application requires to select advisedly the concepts (diagrams, artifacts, etc.) to deal efficiently with analysis and design. In the COOP methodology, we present different criterias affecting the analysis and design phases: to build an application on top of existing ones is one of the provided criterias. These criterias are simple and concrete and require an answer by yes or no in order to adapt the development process. The application context is evaluated *a priori* against these critaria. However, it may sometimes be desirable to change the application context as a result of the development process.

In this paper, we focus on the analysis work when dealing with an application to build on top of existing ones. The analysis work is divided into requirement analysis, domain and business object analysis and system architecture. For each of these steps, we give guidelines related to the different aspects of preservation: functionalities, code and interfaces.

Requirement Analysis

The *requirement analysis* deals with the formalization and organization of explicit requirements (expressed by the user) and implicit requirements (deduced by the analyst). Use cases allow to capture the functional or technical requirements. When building on top of running applications, the requirement analysis helps in identifying the services already provided by the application and the enhancements which will have to be developped.

When functionalities and code are preserved: The focus of requirement analysis is to clearly distinguish the new functionalities from the ones already supported by the running part of the application. With regards to already supported functionalities, a distinction has to be done between functionalities preserved as they are and functionalities enhanced through the new development. Functionalities not directly related to the application under development may also be included, for understanding purpose. Of course, their description do not need to be as detailed as the descriptions dealing with the main functionalities; they can even refer to existing documents (not necessarily written with UML) or, if there is no existing document, directly to the running application. One of the difficulty of the analysis work, in addition to clearly identify and classify the different functionalities, is to integrate all of them in an homogenized way. Therefore, the steps to follow are:

1. Put in the same diagrams preserved and new use-cases according to functional domains. Distinguish them clearly through the kind of stereotypes: <<New>>, <<To-be-modified>>, <<Re-use>>, and <<Out-of-scope>>.
2. Re-organize use-cases included in the same diagram, by using *generalization, inclusion* or *extension* relationships, or by splitting them (especially use-cases describing existing functionalities) in order to allow the homogenization. It is important, while describing a <<To-be-modified>> use-case to let the new functionalities appear explicitly and to link them (through *extension, inclusion* or *generalization* relationships) to the use-case describing the existent functionalities.
3. The split to distinguish the preserved parts from the new ones may lead to diagram(s) difficult to read and functionalities difficult to understand. Therefore complete the diagram with activity diagram(s) to clarify the ordering which are not explicit.
4. Bring together all the <<Out-of-scope>> use-cases in a separate package, also stereotyped <<Out-of-scope>>.

The requirement analysis may lead to changes in stereotypes associated with use-cases. For instance, a use-case stereotyped <<Out-of-scope>> may finally appear as impacted by the new functionalities and therefore stereotyped <<Re-use>>.

A split may also lead to the discovery of <<Out-of-scope>> use-cases. It is important to always have in mind that the goal of the analysis is to isolate the impacted parts of the application from the non-impacted ones, and to clearly distinguish these two families through the use of the <<Re-use>> and <<Out-of-scope>> stereotypes.

When interfaces are preserved: At this stage of the analysis, the focus is on actors using the interfaces. Only actors representing systems (human actors interact with the application through man-machine interfaces, studied in the functional domain aspect) are of interest. It is important to distinguish systems already interacting with the application from systems which will be interacting with it. Indeed, dependent systems do impose constraints which have to be identified during the requirement analysis. The stereotypes presented in our profile help in classifying the different kinds of actors and in highlighting the constraints.

Domain and Business Objects Analysis

The *domain and business object analysis* focuses on the description of the business. Class diagrams and state chart diagrams allow to model the business domain. Enhancements related to the application domain are captured during this step.

When functionalities are preserved: The business description is given through the existing functionalities (instead of being captured from the business domain). The work to be done is similar to what could be done for a new application, but the input of the process is different.

When code is preserved: As during the requirement analysis, our process to deal with preservation is still driven by a clear differentiation between existent and new information.

With regards to the packages stereotyped <<Out-of-scope>> (cf. section 4.1), the steps to follow are:

1. Stereotype all the included classes with <<Out-of-scope>>.

With regards to the other packages, for each class.

1. Stereotype attributes and operations to clearly distinguish the existent from the new ones.
   For each existing element:
   a. if the preservation is qualified by *strong*: use the <<Re-use>> stereotype.
   b. if the preservation is qualified by *medium* or *weak*, choose to modify or not the element. To choose to preserve the element leads to use the <<Re-use>> stereotype. To choose to modify the element consists in stereotyping it with <<To-be-modified>>.
   c. Complete the element description with the characteristics taken from the existent (type, length, and so on.). With regards to operations, modifications have to be documented precisely.
   Stereotype new elements with <<New>>.
2. Generalize the element stereotype to the class if all its belonging elements share a common stereotype. Otherwise, try to isolate the <<Re-use>> elements from the <<To-be-modified>> and the <<New>> elements, by using the *generalization, specialization*, *association* and *composition* relationships, in order to associate a stereotype to the class.
3. Stereotype the associations among classes as done for the elements. The preserved characteristics of the association as well as modifications have to be documented.
4. For each class stereotyped <<Re-use>> or <<To-be-modified>>, if a state-transition diagram is required (the class has a complex behavior which need to be described):
   a. Keep the diagram documenting the running application, or if it does not exist, draw it.
   b. If the class has a behavior different from the one described in the diagram documenting the running application, modify the diagram. It is important to note that a class a-priori stereotyped as <<Re-use>> may finally be modified (and therefore stereotyped <<To-be-modified>>) due to the fact that its state-transition diagram (i.e. its behavior) is changed in the application to be developed.

When interfaces are preserved: The domain and business objects analysis focuses on the description of the constraints related to interactions between the application and the actors especially the actors, stereotyped as <<Constraining system>>. The actor description is completed by the description of classes belonging to the actors. Only classes directly interacting with the application under development are of interest. The idea is to show the services under the responsibility of an external system from the services under the responsibility of the application under development. Therefore:

1. For each actor stereotyped with <<System>>, document its classes directly interacting with the application. Of course, the classes must be linked to the classes of the application under development, otherwise there is no reason to let them appear in the diagrams.
2. Stereotype each class with the name of the actor and document it (component distribution, ...).
3. If required, use activity diagrams to document the interesting steps of the services provided by the actor. For instance, an external component may include a verification which is therefore not required in the current application. Activity diagrams allow to justify why services are or are not supported by the application under development. It may be especially useful to someone not involved in the analysis and willing to participate in the forthcoming phases of the development.
4. Use sequence diagrams to document the interaction between the actors and the application under development.

System Architecture

The analysis of the *system architecture* is a crucial phase. Applications are more and more complex and it is therefore required to think about their logical architecture (documented with component diagrams). Currently, applications are also more often distributed and it is therefore required also to specify their physical architecture (documented with deployment diagrams). When building on top of running applications, this work is even more crucial because in addition to the relationships among the components as well as among the nodes of the application, the relationships with the running components have to be decided.

***When functionalities are preserved*:** It has no impact on this step of the process.

When code is preserved: As through the previous steps, when building on top of running applications, the goal of the system architecture step is to clearly distinguish the existent components from the new ones and to highlight their relationships. We distinguish 4 cases:

- **Case 1:** The running application is preserved and new interfaces are added. It is the case of legacy application, for instance.
- **Case 2:** Modifications on the running application are not very important. It is mostly preserved and new functionalities (easy to isolate) are added. In this case, the existing components are preserved separately from the new components. Existing components use the services of the new components.
- **Case 3:** A lot of new functionalities are integrated in the application. They can't be isolated from the existing components. This situation is called *re-design*.
- **Case 4:** The existing architecture is preserved and modifications (not very important) are integrated in it.

The Figure 2 summarizes the different cases. Note that case 1 and 2 may coexist.

Each case presented previously need a suitable architecture. With regards to *case 1* and *2*, the steps to follow to deal with such an architecture are:

1. Organize the classes stereotyped with <<Re-use>> into components, stereotyped also with <<Re-use>>. In *case 1*, the stereotype qualifying the running components may even be better specified, for instance with <<Encapsulated>>. If the application under study is a legacy application, the stereotype may be named <<Legacy>> instead of <<Encapsulated>> to emphasize the application context.
2. Organize the classes stereotyped <<New>> into components.
3. Identify the links among the components, especially between components stereotyped <<New>> and <<Re-use>>.
   In *case 1*, check that:
   a. actors interact only with <<New>> components,
   b. relationships among <<Re-use>> components and <<New>> components always indicate that <<New>> components use the services of the <<Re-used>> components (and not the contrary).
   In *case 2*, on the contrary, check that:
   a. actors interact only with <<Re-use>> components
   b. relationships among <<Re-use>> components and <<New>> components always indicate that <<Re-use>> components use the services of <<New>> components (and not the contrary).

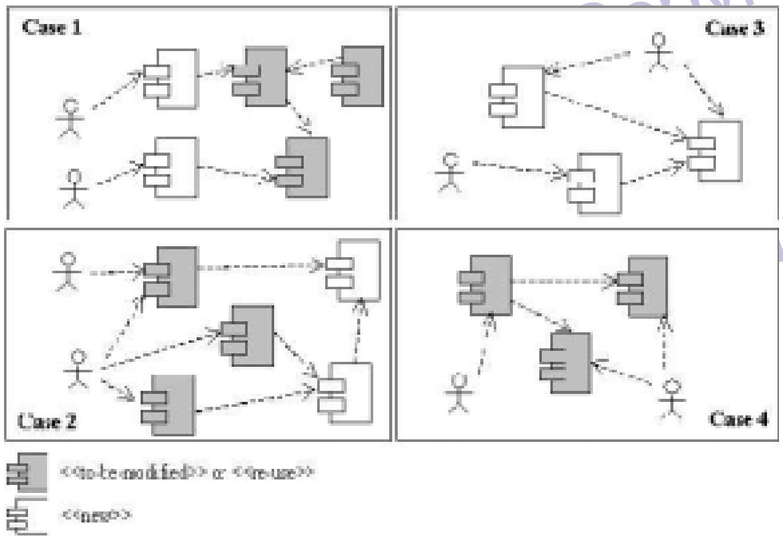
In *case 3*, the whole architecture is re-designed as if there was no existent to start from. In this case, it is useless to indicate at the component level if it includes classes stereotyped as <<New>>, <<To-be-modified>> or <<Re-use>>, because all the components will have to be created (from existing code or not) and will therefore be considered as <<New>> ones.

In *case 4*, the steps to follow are:

1. Build the component diagram from the documentation related to the running application. If there is no documentation, the component diagram has to reflect the current architecture and not the whished one.
2. Associate the classes <<To-be-modified>> and <<Re-use>> to their components.
3. Place the classes stereotyped <<New>> into existing components. This placement has to be driven by the relationships among the classes under study. This work may lead to the conclusion that the architecture is not suitable and that it has to be changed (cf. *Case 2*).
4. Stereotype the components which require enhancement (through classes <<New>> or <<To-be-modified>>) with <<To-be-modified>> to highlight what will have to be taken under consideration through the development.

When interfaces are preserved: There is no particular work to do in addition to the standard analysis process. The component diagram(s) is(are) built as for application developped from scratch.

Figure 2: Component architecture



CONCLUSION

In this paper we presented a UML profile for building on top of existing applications. We do not reduce the existing application to its code only. We also take into consideration the expertise about the functional domain and the interfaces describing the relationships the running application may have with others systems. The preservation is also qualified by *strong*, *medium* or *weak*.

The profile presented in this paper is part of the COOP methodology (Contextual Object-Oriented Process) where we propose a flexible approach for analysis and design with regards to the context of the application. The profile is used through the fragments of the methodology dedicated to applications developped on top of running ones. By situating the application in its context, by choosing the interesting fragments and using the dedicated profile, the development process is improved.

In the future, we would like to improve the COOP methodology and the associated profile to better highlight the incrementality of the process, and to handle traceability through the different steps and fragments of the methodology as well as through the different kinds of UML diagrams used during the development process.

REFERENCES

[Be97] K. Beck, Smalltalk Best Practice Patterns, Prentice Hall, Englewood Cliffs, NJ, 1997.

[BRJ98] Booch G., Rumbaugh J., Jacobson I., The Unified Modeling Language User Guide, Addison-Wesley, 1998, Object Technology Series.

[Fo99] M. Fowler Refactoring : Improving the Design of Existing Code, Addison-Wesley, 1999, Object Technology Series.

[IWP99] E. Insfran, R. Wieringa, O. Pastor, Using TRADE to improve an object-oriented method, University of Twente, 1999.

[JBR98] Jacobson I., Booch G., Rumbaugh J., Unified Software Development Process, Addison-Wesley, 1998, Object Technology Series.

[Kr00] P. Krutchen, The Rational Unified Process. Addison-Wesley, 2000, Object Technology Series.

[KS00] M. Macona Kandé, A. Strohmeier, Towards a UML Profile for Software Architecture Descriptions, UML 2000: 513-527
[NuF00] N. J. Nunes, J. Falcão e Cunha, Towards a UML profile for interaction design: the Wisdom approach, UML 2000: 101-116
[OMG] Object Management Group, http://www.omg.org/.
[Op92] William Opdyke, Refactoring Object-Oriented Frameworks, PhD Thesis, Illinois 1992.
[Ral01] J. Ralyte, Ingenierie des methodes a base de composants. PhD Thesis. Universite Paris I – Sorbonne. January 2001.
[RJB98] Rumbaugh J., Jacobson I., Booch G., The Unified Modeling Language Reference Manual, Addison-Wesley, 1998, Object Technology Series.
[RR01] J. Ralyte, C. Rolland. An Assembly Process Model for Method Engineering. CAISE 2001, p. 267-283. June 2001.
[SH96] K. van Slooten, B. Hodes. Characterizing IS Development Projects. IFIP TC8, WG 8.1/8.2, p. 29-44. August 1996.
[Sou98] D. D'Souza, Catalysis: Objects, Components, and Frameworks with UML, Addison-Wesley, 1998, Object Technology Series.