Adding Use Cases to the building of Information Systems with Oracle Designer

Philippe Saliou, Vincent Ribaud

Adding Use Cases to the building of Information Systems with Oracle Designer

P. Saliou and V.Ribaud
EA2215, Département d'Informatique, Faculté des sciences, 29285 Brest Cedex, France
E-mail:{philippe.saliou,vincent.ribaud}@univ-brest.fr

Abstract

The purpose of the work described in this article is to introduce UML use cases in a traditional development process of medium and small sized Information Systems. This work relies on the CASE tool Designer from the Oracle company and its associated development method CADM.

This introduction has an impact on the development process used as well as on the tools notably in terms of coupling, transformation and traceability of the various models used.

The three axis of this work are the following ones :

- Adding a phase of requirements capture with the help of use cases prior to the CADM development method currently used with Designer.
- Defining rules to draft the use cases as well as rules for the transformation and traceability towards Designer analysis and design models (function hierarchy, design modules).
- Extending the CASE tool and quite especially its repository so that it integrates the management of use cases: general properties, traceability links between the various elements contained in the repository, versioning, etc.

1. Introduction

Methods and tools are essential for the achievement of a project. The CASE tools are often confined to the analysis and conception phases while the 4GL environments are used for the implementation. Oracle Designer is amongst the rare CASE tools which can be used at each phase of an information system development cycle.

We have employed Designer for several industrial projects according to the CADM waterfall process. Within the framework of a client-supplier contracting relationship, it has always been necessary to complement the Designer models with requirements specifications. The use cases are appropriate to express most of the functional requirements. It seems natural to complement the CADM method with a requirements capture phase relying on use cases.

During the analysis, processings are represented in Designer as a function hierarchy. The use cases model can be organized with the help of packages and relations. This organization makes easier the transformation into an initial function hierarchy.

One of the Designer main components is its repository which stores all information concerning the system under construction. Its extension capacity allows the integration of the use cases model in the repository, linked with the other Designer models. This makes easier project management, collaborative work, requirements traceability, etc.

The article is as follows. Section 2 concerns requirements management in the CADM method. Section 3 presents use cases coupling with the CADM method. Section 4 describes use cases integration in Designer. Section 5 summarizes the prospects and is followed by a conclusion.

2. How are requirements managed in Designer ?

Designer and the associated CADM method

Oracle Designer is an extremely powerful integrated CASE tool. It allows the whole building of an Information System all along the phases of the software life cycle. To this end, it relies on a unique common repository stored in an Oracle database.

Our development approach relies on CADM (CASE Application Development Method), a waterfall process (Analysis->Design->Build->Implement->Production)) by Paul Dorsey and Peter Koletzke [1]. This approach is a derivative of the Case*Method by Richard Barker [2].

CADM belongs to the family of systemic methods. The data and processings have first to be separately modelized, and then coupled to constitute a unique and integrated system. The building of the system gets through different abstraction levels: analysis, design and implementation.

Requirements capture and function hierarchy

There is no requirements capture model in Designer.

<u>During the preliminary analysis phase,</u> the requirements capture materializes mostly in a textual form casually laid out or structured by a requirements plan.

<u>During the general analysis phase,</u> the requirements capture becomes elaborate, detailed and reshaped through a function hierarchy and an Entity/Relationship data model.

The function hierarchy is the model proposed by Designer to analyze processings. Processings in the information system are hierarchically divided into a set of activities known as functions. Therefore a function is a more or less important activity which can be automatized or manual. Some functions can be shared, in that case they appear several times in the hierarchy with a distinctive sign.

From this phase, a particular attention allows to identify the functions which will map the application modules. The requirements capture is quite rarely structured, so it is not possible to map the functions and entities obtained throughout this phase. This makes all the more sense as the requirements capture is exterior to Designer. ***Very often the function hierarchy and the E/R model are used as elements of work and discussion with the users to validate and approve requirements.***

<u>During the detailed analysis phase,</u> the function hierarchy is refined and completed: entities usages are defined for every function as well as attributes usages while cross-reference controls are performed between data and functions.

3. Coupling of use cases and CADM method

Why introducing a new form of requirements capture?

The requirements capture is still one of the most difficult task in the software development, the one that has without doubt the strongest impact on the success or failure of a project.

In our development approach, during the analysis phase, the requirements capture materializes as a function hierarchy. Through this approach, the point of view is the one of the system: what are the functions that the system must offer to fulfil the final user needs?

In the use cases, the point of view is the one of the system users: what do the various users expect from the system, what are their aims ? The use cases solve the drawbacks of hierarchical functional decomposition in terms of capture and understanding of requirements. Nevertheless, we have to cope with the transformation of these use cases into models of analysis and conception.

UML and the Unified Process [3] have made popular the idea of complementing the analysis models with the use cases. This led us to bridge the two points of view:

- <u>An external point of view</u> through the use cases for the requirements capture.
- <u>An internal point of view</u> through a function hierarchy relying on an Entity/Relationship model.

For the building of medium and small sized systems, the fact of taking into account the use cases has an effect only on the tasks and deliverables linked with the requirements capture.

- As the requirements capture relies on a better structuration, the UML model of the use cases may be integrated directly into the Designer repository.
- It is possible to transform automatically an external point of view (use cases) into an internal point of view (function hierarchy), by relying on the organization of use cases in terms of relations as well as functional regroupment in packages.
- The various use cases map the hierarchy functions. This traceability permits to reduce the textual description of functions and helps to measure the impact of changes in requirements.

Organization of use cases

UML allows an organization of the use cases with the help of relations and packages.

The relation of inclusion allows a use case to explicitly insert another use case, in a specified place of its scenarios. This relation is used to avoid the repeated description of the same serie, by “factorizing” the common behaviour in a use case set apart. This notion of inclusion also allows to materialize the notion of “sub-case” described by Cokburn [4].

The relation of extension allows a use case to implicitly insert another use case, in an optional way, in a place indirectly specified in the one which operates the extension. This relation is used to separate an optional or rare behaviour from the mandatory behaviour.

The relation of generalization allows a use case to inherit the description of a parent use case. This relation is used to formalize important variations on a same use case.

The package allows the gathering of UML elements into coherent sets.

Several strategies are available to operate a regroupment of use cases: by actor, by functional domain, etc. Our aim is to reach a functional decomposition (function hierarchy), so we use a ***strategy of regroupment by functional domain***.
The included use cases of “factorization” type may be regrouped in a particular package, viewed as a common support service, to distinguish it effectively from the functional cases which include it.
An included use case of “sub-case” type is likely to find itself in the same package as the calling use case, both of them belonging likely to the same functional domain.

Transformation rules of use cases into functions

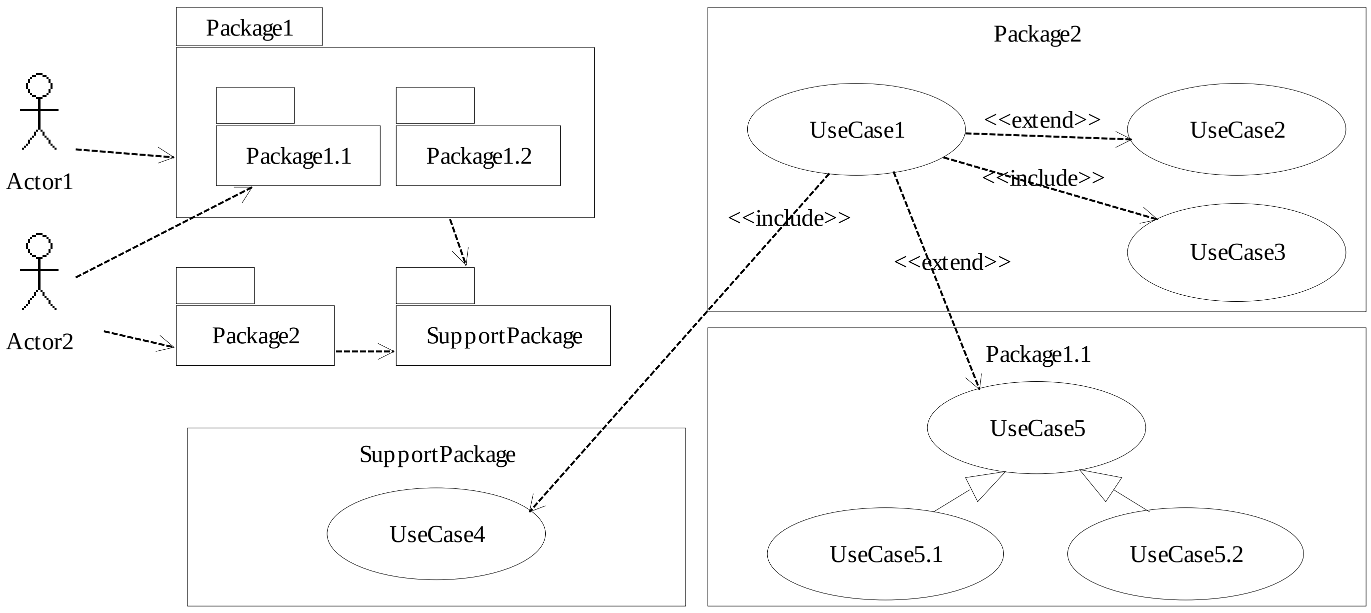
A function hierarchy by default proceeds from the organization of the use cases in terms of relations and packages. The transformation rules are as follows:

- The root function corresponds to the system.
- Each package or sub-package gives birth to an intermediary function attached to the root function (if it is a package) or to the function stemming from the parent package (if it is a sub-package).
- Each use case gives birth to a function:
    - Intermediary function if the use case includes other use cases or if it specializes in other use cases.
    - Function attached
        - To the function stemming from the including use case if a relation of inclusion is involved.
        - To the function stemming from the parent use case if a relation of specialization is involved.
        - To the function stemming from the package that contains the extended use case if a relation of extension is involved.
        - To the function stemming from the package that contains the use case, in all other cases.
    - A use case of “factorization“ type gives birth to a function shared by several branches of the hierarchy.
    - A use case which is an extension of other use cases may also give birth to an function shared by several branches of the hierarchy. This will occur if the extended use cases belong to different packages.

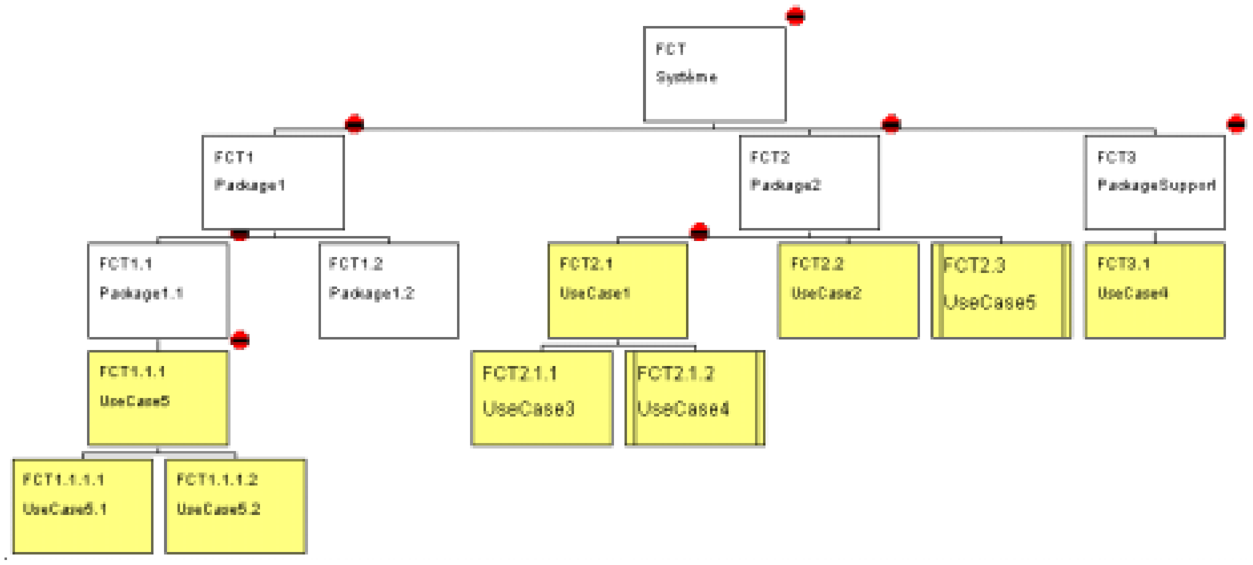
At the end of the analysis, Designer allows the transformation of the analysis model (functions and their entities usages) into a rough draft of design model (modules and their tables usages). This explains why the function hierarchy obtained by default is refined, not only in terms of functions provided with the system, but also in the perspective of its future transformation in design modules.

At the end of this refining process, some use cases will find themselves dispatched as several functions while others will be regrouped partially or totally in a unique function. In such an approach, it is important to maintain the bidirectional links between the functions and the use cases.

An example of transforming use cases into function hierarchy



Functions issued from use cases are grey and shared functions are bordered by double lines.



4. Use cases integration in Designer

Use cases integration is operated in Designer by the way of extensions added to the repository. These extensions allow the storage in the repository of all documents and elements related to the use cases as well as the setting up of bidirectional traceability mechanisms between the actors, the use cases and the function hierarchy.

Henceforth the use cases management is totally monitored by Designer, either rights management and resources sharing, versioning, archiving, measuring impact of change or even document searching.

Use cases management

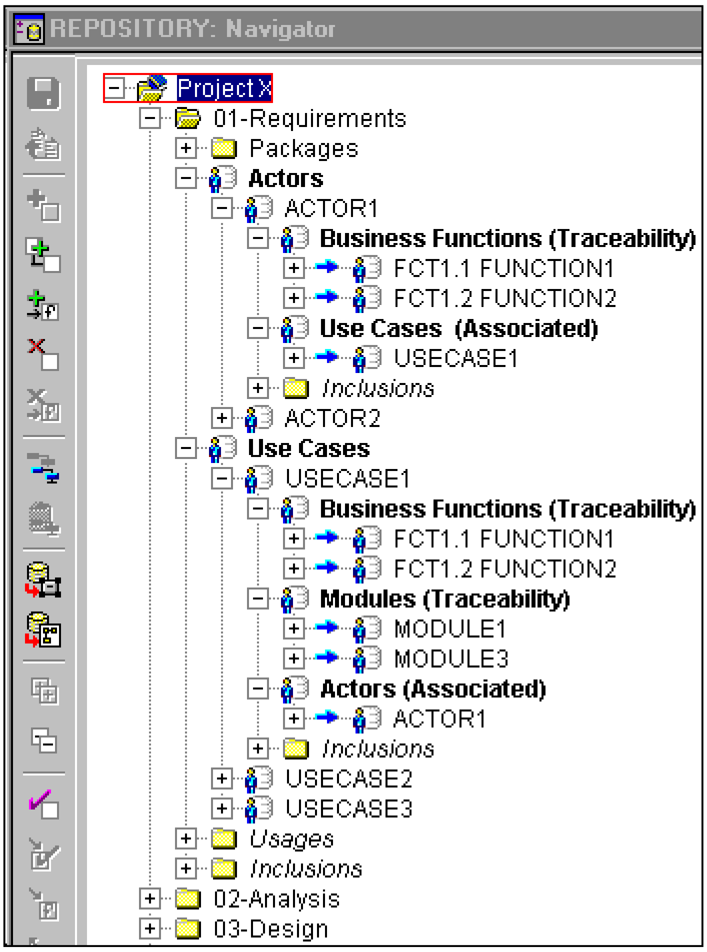
The main extensions added to the Designer repository materialize as a framework consistent with the UML use cases meta-model. This framework is complemented to allow:

- The storage in the repository (thus included in a database) of the use cases scenarios as documents issued from various word processors.
- The mapping of each UML element with one or several hypertext links towards documents stored in the repository, allowing their consultation and upgrading from Designer.
- The availability of extra features exceeding those provided by UML and particularly additional project management information.
- The easy manipulation and consultation of the links between the various UML repository elements: actors, use cases and packages.

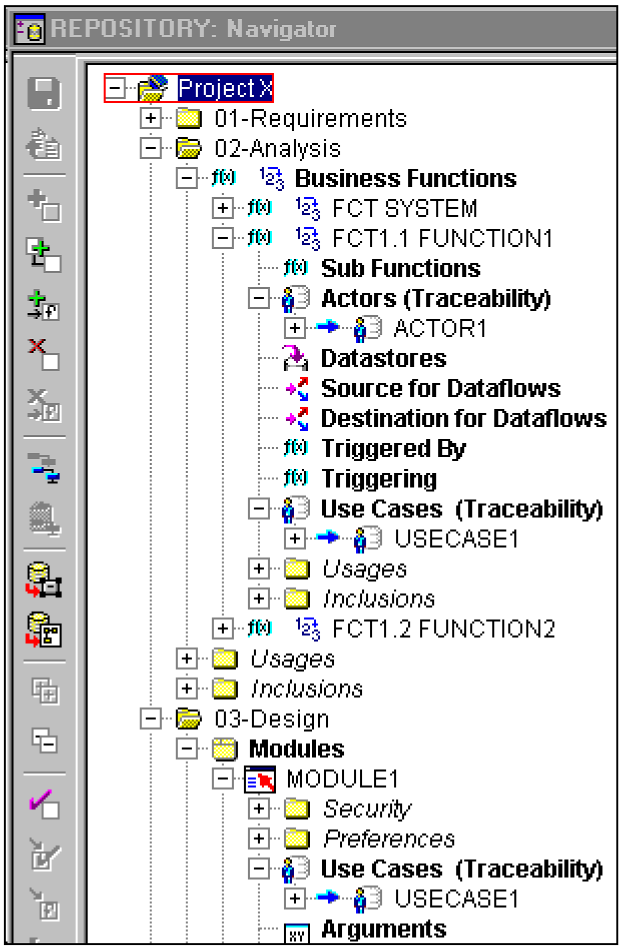
Mapping between use cases and other models

The extensions added to the Designer repository allow to map the requirements represented by the use cases with the functions and all other elements of analysis and conception existing in the repository. This mapping allows to perform cross-reference checks in order to ensure, for example, that the conception covers the whole requirements.

The screen captures below shows both directions of this traceability: from the use cases towards the other models and from the other models towards the use cases.



```
REPOSITORY: Navigator
Project X
  01-Requirements
    Packages
    Actors
      ACTOR1
        Business Functions (Traceability)
          FCT1.1 FUNCTION1
          FCT1.2 FUNCTION2
        Use Cases  (Associated)
          USECASE1
        Inclusions
      ACTOR2
    Use Cases
      USECASE1
        Business Functions (Traceability)
          FCT1.1 FUNCTION1
          FCT1.2 FUNCTION2
        Modules (Traceability)
          MODULE1
          MODULE3
        Actors (Associated)
          ACTOR1
        Inclusions
      USECASE2
      USECASE3
    Usages
    Inclusions
  02-Analysis
  03-Design
```



```
REPOSITORY: Navigator
Project X
  01-Requirements
  02-Analysis
    Business Functions
      FCT SYSTEM
      FCT1.1 FUNCTION1
        Sub Functions
        Actors (Traceability)
          ACTOR1
        Datastores
        Source for Dataflows
        Destination for Dataflows
        Triggered By
        Triggering
        Use Cases  (Traceability)
          USECASE1
        Usages
        Inclusions
      FCT1.2 FUNCTION2
    Usages
    Inclusions
  03-Design
    Modules
      MODULE1
        Security
        Preferences
        Use Cases  (Traceability)
          USECASE1
        Arguments
```

5. Related work and prospects

This work allows the integration of use cases management in a model-driven development tool. The same problematic is addressed and resolved by the integration of Rational RequisitePro and Rational Rose. However, the Rational Suite Development is UML-based, when Designer is dedicated to the Information System through its own models (entity-relationship, function hierarchy, etc).

This work is influenced by the Unified Process approach. In this perspective, the use cases are not a mere specification tool for the system requirements. They also guide the design, the implementation and the tests [3].
We also forecast to monitor the testing process through the use cases, keeping the automation and traceability targets previously described in this article.

Designer is an extremely powerful tool which, as a counterpart, enjoins quite a strict development frame. Designer usage is based on modelization and models transformation belonging to various abstraction levels: analysis, conception, implementation. While one works at a given level, it is necessary to be well acquainted with the inferior level possibilities as well as the transformation rules of the models in this level. Practising the tool and understanding the way it works leads to use one level possibilities in the expectation of what will be generated at the following levels. This allows to work at a high abstraction level and to take advantage of a considerable leverage on the generated code.
Adding and integrating use cases into the CADM method and Designer provides an additional abstraction level describing requirements. With more pratice, we shall improve the writing and structuring rules of use cases. We will then take full advantage of this extra level to build even better systems in a quicker wa*y.*

6. Conclusion

This paper has presented how to add use cases to the building of Information Systems with Designer, according to three main directions: adding a phase of requirements capture, defining rules for drafting and transforming, managing use cases in the repository.

The first benefit for a firm is that the power and ability (in terms of technique and method) of a CASE Tool such as Designer are preserved while taking advantage of some aspects of UML and Unified Process. Such an approach could allow a progressive and justified transition from traditional development to UML for firm developers [5]. This transition will be made easier because UML will be introduced through the Unified Process and not by the modelling language

7. Références

[1] Paul Dorsey and Peter Koletzke, Designer/2000 Handbook, Oracle Press, 1997.

[2] Richard Barker, Case Method: Tasks and Deliverables, Addison-Wesley Longman, 1990.

[3] Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified Software Development Process, Addison-Wesley Longman, 1999.

[4] Alistair Cockburn, Writing Effective Use Cases, Addison-Wesley Longman, 2001.

[5] H. Lindsey, P.R. Hoffman, Bridging traditional and object technologogies: Creating transitional applicartion , IBM Systems Journal. Vol 36 N°1, 1997.