

LABORATOIRE



INFORMATIQUE, SIGNAUX ET SYSTÈMES  
DE SOPHIA ANTIPOLIS  
UMR 6070

# JECKO, A CONTEXT-DRIVEN APPROACH TO ANALYSIS AND DESIGN

*Isabelle Mirbel, Violaine de Rivieres*

*Projet MECOSI*

Rapport de recherche  
I3S/RR-2002-03-FR

février 2002

---

RÉSUMÉ :

MOTS CLÉS :

---

**ABSTRACT:** In order to address the problem of adapting methodology to specific environment, we propose JECKO, a context-driven approach to analysis and design. In this approach, the software context is specified through four basic criteria. We show how we could take advantage of the software context through a flexible framework that supports analysis and design. The adaptability is handled through fragmentation. Each fragment is provided with a generalized description of modeling rules and/or guidelines. We distinguish prime fragments, useful whatever the software context might be, from specific fragments, that are dedicated to a particular context. Modeling rules and guidelines are illustrated with the UML notation. In order to tailor the development process for a specific software, method fragments are selected with regards to the particular software context. The chosen fragments define the route map, dedicated to the software that is being considered. By following the route map adapted to the software context, the analysis and design activities, as well as their outcomes, could be enhanced and better controlled.

KEY WORDS :

# JECKO, a Context-Driven Approach to Analysis and Design

Isabelle Mirbel  
Laboratoire I3S  
Route des Lucioles - BP 121  
06903 Sophia Antipolis Cedex  
France  
email: Isabelle.Mirbel@unice.fr

Violaine de Rivieres  
Amadeus sas  
485 Route du Pin Montard, B.P. 69  
06902 Sophia Antipolis Cedex  
France  
email: vrebuffel@amadeus.net

## Abstract

*In order to address the problem of adapting methodology to specific environment, we propose JECKO, a context-driven approach to analysis and design. In this approach, the software context is specified through four basic criteria. We show how we could take advantage of the software context through a flexible framework that supports analysis and design. The adaptability is handled through fragmentation. Each fragment is provided with a generalized description of modeling rules and/or guidelines. We distinguish prime fragments, useful whatever the software context might be, from specific fragments, that are dedicated to a particular context. Modeling rules and guidelines are illustrated with the UML notation. In order to tailor the development process for a specific software, method fragments are selected with regards to the particular software context. The chosen fragments define the route map, dedicated to the software that is being considered. By following the route map adapted to the software context, the analysis and design activities, as well as their outcomes, could be enhanced and better controlled.*

## 1. Introduction

Simple methodology that has proved its power for certain types of software development may be quite unsuitable for others. There is no universal methodology [6]. Therefore it seems to be a need for processes that are flexible and adaptable to different kinds of software development. The need for situation-specific approaches, to better satisfy particular situation requirements, has already been emphasized [10, 9, 7]. In order to address the problem of adapting the methodology to its specific environment, we propose JECKO, a context-driven approach to analysis and design.

Our work focuses on the analysis and design phases of the software development process (SDP). Going through a situation-driven approach means first to understand the software context to then tailor the analysis and design process (A&DPr) to the specific software in order to control and improve the development. To specify the software context as well as to provide a customisable A&DPr is not straightforward. In JECKO, the software context is tackled through four basic criterias:

1. How to deal with running software ?
2. How to deal with graphical user interface?
3. How to deal with database?
4. How to deal with distributed software?

By situating the software under consideration with regards to these criterias, the software context is specified.

Then, we show how we could take advantage of the software context through a flexible framework that supports analysis and design. The adaptability of the method is handled through fragmentation. Each fragment is provided with a generalized description of modeling rules and/or guidelines. We distinguish prime fragments, useful whatever the software context might be, from specific fragments, that are dedicated to a particular context.

Most of the time, methodologies are imprecise, too much general and not well-adapted to specific problems [6]. In JECKO, precise guidelines are given and illustrated through target notation: UML [1] has been chosen as it is world-wide standard.

In order to tailor the development process to specific software, method fragments are selected with regards to the particular software context. The chosen fragments define the route map, dedicated to the software that is being considered. By following this route map adapted to the software context, the analysis and

design activities (A&DAc), as well as their outcomes, could be enhanced and better controlled.

In this paper, we first present the criterias to specify the software context (Section 2). Then we explain how flexibility is handled through fragments (Section 3). Afterwards focus is made on the requirement analysis phase (Section 4), and finally we conclude in Section 5.

## 2. The software context

Software development success requires initially the advised selection of modeling concepts (model elements, diagrams, artifacts, etc.) in order to deal efficiently with A&DAc. In this section, we present four criterias to specify the software context. Thanks to these criterias, the A&DPr can be better adapted by selecting tasks to do and artifacts to use for getting accurate outcomes.

### 2.1. Dealing with running software

Most of the softwares are now built on top of running software. In this context, it is required to specify precisely what has to be kept from what has to be replaced or enhanced. To handle correctly such preservation, the analysis activity has to focus on the interfaces with the exiting software; and boundaries between both software pieces, new and running one, have to be precisely identified and documented. Dedicated A&DPr is therefore suitable.

Different aspects of running software may be taken into consideration. In addition to the *code* itself, the expertise about the *functional domain* (taken from the functionalities, data and screen shots) may also be of interest. The *interfaces* describing the relationships the running software has with other systems (softwares, databases, ...) should also be taken into consideration. These three aspects have to be distinguished because a given project could focus only on one aspect. For instance, if the project starts from Rapid Application Development (RAD), only functional domain aspect should be kept from the running software. Keeping the code is different from keeping the functionalities and/or from keeping the interfaces. In consequence, the A&DAc will be different with regards of the preservation kind. Different diagrams and modeling concepts may be more or less suitable to highlight different reuse aspects. It will also have an impact on the forthcoming phases of the SDP. It is important to early distinguish these three aspects (functional domain, interface, code) in the A&DAp.

The analysis phase is dedicated to clearly identify *what* has to be preserved, while the design phase focuses on *how* such preservation can be handled. To carry on properly the design activity, how the running software will be taken into consideration has to be specified precisely:

- it can be taken as it is; for instance, while interfacing an running component without any possibility to modify it (this is the case of bought component);
- or it can be slightly modified; for instance, while interfacing an running component, developed by the company, but used by other company softwares;
- or finally, it can be widely modified; for instance, when software evolves into new version and the development team is the owner of the running part.

These three preservation aspects (functionalities, interfaces, code) should be modulated in order to better be exploited during the development phase. 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 with few limits. Preservation specification and qualification help to tailor the A&DPr to manage it more efficiently. In [5], the authors propose a profile to deal with such context through the UML notation.

### 2.2. Dealing with graphical user interface (GUI)

Lot of tools and techniques are currently available on the market and propose generic frameworks to facilitate GUI development, reducing the design activity for this aspect. In consequence to fully benefit from such tools and techniques, the A&DPr has to be adapted by isolating the GUI specification from the business one; this is not so easy for end-users who are usually enclined to describe the software business only through its GUI. Therefore, an adaptation of the A&DAp to facilitate this distinction between the GUI and the business enhances A&DPr.

### 2.3. Dealing with database

Dealing with database requires to organize data in such way that specific database rules (as the normal form decomposition one) can be followed. Traditional A&DPr [3, 2] propose efficient ways for dealing with database design. The A&DPr should take advantage of such traditional techniques.

Moreover, dedicated concepts are required to build effective database: primary key, index, etc. To fully manage these concepts, additional information has to be captured through the analysis activity.

And finally, if the project under development includes also the choice of one database management system, the analysis outcome must be one key input for this choice.

For these three reasons, the A&DPr has to be adapted with regards to database aspect.

## 2.4. Dealing with distributed software

Distributed softwares are more complex to apprehend. Most of the time, they require particular organization of the A&DAc to fully handle the distribution aspects. Moreover, specific problems and requirements may be encountered with distributed software (link feature, speed, network security aspects, ...). The A&DPr has to be adapted to handle correctly such particularities.

The four criterias we have presented are simple and concrete. They only require an answer by *yes* or *no* to allow the A&DPr adaptation and to improve its outcomes. The software context is evaluated *a priori* with regards to these criterias.

## 3. The Analysis and Design Framework

### 3.1. Analysis and Design Activities

The A&DPr proposed in JECKO starts with the *Requirement Analysis* to formalize requirements. It covers explicit requirements, expressed by the end-user, as well as implicit ones, deduced by the analyst. In addition to functional requirements, technical ones have also to be considered carefully. Their pertinent organization facilitates the progress of the forthcoming phases.

The second phase, *Domain and Business Object Analysis*, focuses on the specification of the business covered by the software to be developed. This is crucial phase of the analysis activity to get software fulfilling the requirements previously captured and to anticipate future enhancements not yet fully identified.

*Requirement Analysis* and *Domain and Business Object Analysis* phases may be processed in parallel. They provide two complementary and essential view points on the software to be developed.

The third phase concentrates on the *Software and System Architecture*. Softwares are more and more complex from the functional and technical points of view. Most of the time, software involves heterogeneous environments, which increase considerably the complexity of the architecture to be deployed.

Then, the fourth phase, *Component Specification*, copes with the integration of the different components through the identification of the interfaces. Furthermore, potential reuse aspects have to be handled while defining the component interfaces.

Finally, through the *Internal Design*, the specification of each element (component, class, and so on) of the software is refined.

The JECKO framework supports standard decomposition into the previously-defined phases, in order to be used by anyone without fundamentally changing his main process phases. The originality of our work consists in the fragmentation proposed to handle flexibility; and especially the way fragments are organized to facilitate customization of any A&DPr with regards to the software context.

### 3.2. Handling Flexibility

To support flexibility, A&DAc are decomposed into atomic tasks. Each of them has to be situated with regards to the others and its adequacy to the software context to allow the A&DPr adaptation. Inside phases, tasks are organized according to their role within the activity: Four main steps come to view whatever is the phase under consideration.

- The first step, *Introduction Step*, copes with preliminary tasks to enhance the forthcoming step progress.
- The second step, *Core Step*, encompasses the main tasks of the current phase. These tasks concentrate on the elaboration of the phase outcome.
- The third step, *Refinement Step*, focuses on organizing and refining the various diagrams and concepts built through the previous steps.
- The last step, *Investigation Step*, concerns the specific tasks required to better handle context-related additional aspects of A&DAc. They appear to be crucial for the A&DAc success.

This generic skeleton is applied through each phase of the JECKO analysis and design framework, as it is shown in Figure 1.

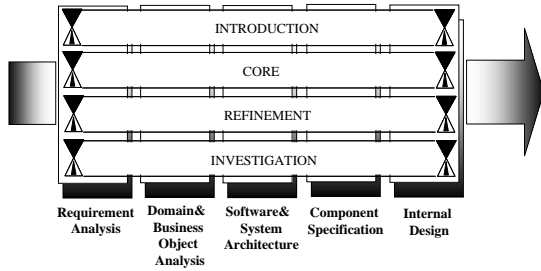


Figure 1. JECKO Analysis and Design Framework

Analysis and design tasks mainly correspond to guidelines and modeling rules. In JECKO, tasks are presented in terms of fragments where their situation within the A&DPr is given in addition to the guidelines.

Depending on the software context, fragments will be selected to define the route map of the software under consideration. Route map is an A&DPr instance to answer the specific needs of the software under development.

### 3.3. Embedding guidelines into fragment

The A&DPr has to be adapted, but whatever is the software context, A&DAc include unavoidable tasks: specification of the system boundaries, business description, etc. These tasks have to be handled differently depending on the software context. Furthermore, context-related additional tasks may also be suitable through the A&DPr.

Therefore, in JECKO, we propose a fragment-based approach to provide a generic and adaptable way of handling the A&DPr tasks. We distinguish fragments dedicated to a particular criteria, from context-independent fragments. They are named respectively prime fragments and dedicated fragments.

A fragment is composed of:

- **Name:** to identify the fragment;
- **Situation:** to position the fragment within the framework and with regard to the JECKO criterias;
- **Intention:** to clarify the purpose of the fragment;

- **Guidelines:** to explain how to answer the intention. Guidelines are illustrated through the UML notation [4, 8, 1] which is now a standard notation used by programmers as well as domain expert, through the entire development cycle.

- **Associated fragments:** to point out related fragments which have to be considered when defining the route map because their *intentions* share some aspects with the current fragment.

In the JECKO framework, we have defined the fragments for the five analysis and design phases presented above. In this paper, we focus on the *Requirement Analysis* phase.

## 4. Handling flexibility through requirement analysis

Requirement analysis phase main goal is the identification of the critical points and the boundaries of the software. Guidelines corresponding to these main tasks are given in the *Core step*. Depending on the software under development, especially for distributed softwares, the requirement analysis activity has to start by the identification of the different software pieces; this is handled during the *Introduction step*. The different diagrams resulting from the introduction and core steps may then need to be reorganized depending on the software context in the *Refinement step*. For instance, softwares including database require additional dedicated work to highlight where persistence is needed. Finally, in the *Investigation step* we propose guidelines and recommendation to manage the additional information related to the requirements. For instance, sequencing among the different diagrams may be provided to show the interaction between them, error and constraints may be taken into account.

Figure 2 shows how the flexibility is handled through each step of the requirement analysis phase. Prime fragments are represented with bold line; dedicated ones are mainly differentiated through their icon. Fragments represented on the same horizontal line can be applied in parallel; on the contrary, fragments appearing on the top of the figure have to be applied before fragments appearing at the bottom of the figure.

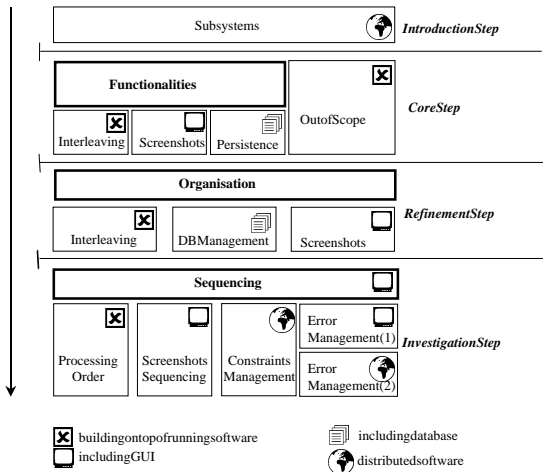


Figure 2. Requirement Analysis Framework

#### 4.1. Dealing with running software

Building on top of running software means to document very precisely the interleaving between the elements to be kept in the running software and the new ones. The key point of requirement analysis is to clearly distinguish new functionalities from existing ones supported by the current running system. With regards to already supported functionalities, clear distinction has to be done between functionalities preserved as they are, and enhanced ones by the new development. Therefore dedicated fragments, called *Interleaving* (their descriptions are given in the appendix), are provided in the *Core step* and to the *Refinement step*.

Functionalities of the running software, which are not directly related to the new development may also be included, for documentation and understandability purposes. Of course, their description do not need to be as detailed as the ones of the enhanced and new functionalities; they can even refer to existing documents (not necessarily written with UML) or, when there is none, directly to the running software. Dedicated fragment, called *Out of scope* let to manage these documentation tasks.

#### 4.2. Dealing with graphical user interface

When the software includes a GUI, it is also necessary to give precise information on its screen shots. It is the purpose of the *Screenshots* (refer to the appendix) and *Screenshots sequencing* fragments.

#### 4.3. Dealing with database

When the software includes database, dedicated tasks have to be done through the requirement analysis. They are presented in the *DB Management* and *Persistency* fragments (refer to the appendix).

Furthermore, error management has to be studied very carefully when the software includes GUI and/or is distributed, this is objective of the *Error Management* fragments.

#### 4.4. Dealing with distributed software

Finally, distributed software needs dedicated tasks to split the software into different physical entities (commonly named subsystem) and to specify the constraints related to the software distribution. The *Subsystems* and *Constraint Management* fragments handle these particular aspects (refer to the appendix).

Thanks to the software context given through the four criterias, the *Requirement Analysis* phase framework can be tailored for the software specificity. Route map is defined through the mandatory *Prime fragments* and the selected *Specific fragments*. Route map example is shown in Figure 3: it corresponds to one JECKO route map for distributed software with database and GUI.

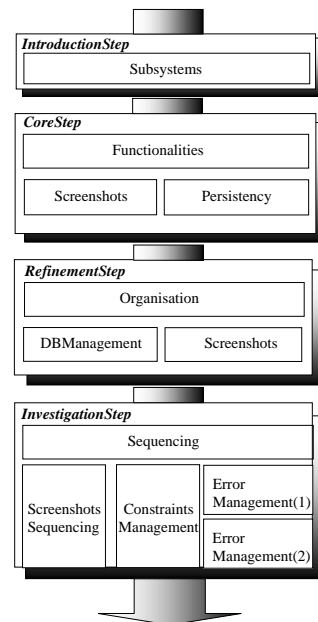


Figure 3. Requirement analysis - An example of route map

Following the route map through JECKO A&DPr, the focus is really put on the software key points and on its particularities in order to enhance the A&DAc quality as well as the outcomes of this first crucial SDP phase.

## 5. Conclusion and Perspectives

In this paper, we have presented JECKO, a context-driven approach to analysis and design. Four basic criterias from which the software context is specified are proposed. We have explained how to take advantage of this context specificity through a flexible framework to support A&DAc. Flexibility is handled through fragments, which provide a generalized description of modeling rules or guidelines. Prime fragments, useful whatever the software context is, are distinguished from dedicated ones, customized to answer to specific context. Fragments are selected with regards to the software context, defining route maps to be followed through the A&DPr. In this paper, we have presented the *Requirement Analysis* phase more in detail. Pertinent fragments have been extracted to illustrate this phase of the framework and, more generally, the context-driven approach provided by JECKO.

With regards to the criteria used to tailor the A&DPr, our work focuses on information about the software to be developed. In the future, we would like to include information about the project [9], as for instance, the time pressure or the dependency with other projects.

Route maps through the A&DPr are deduced from the context of the software under development, with regards to determining criterias. In the future, we would like to improve our framework by weighting the route maps with regards to the designer expertise. Depending on the person in charge of the A&DAc (junior or senior analyst, business or technical one) the diagrams/concepts which are the most suitable, the most useful to help through the A&DPr process may be slightly different.

From the requirement analysis to the internal design, it is important to follow the evolution of the different elements of the modeling under consideration, in order to be able to justify and to explain the forthcoming stages of the SDP, but also to improve the quality of the development [11]. Therefore, we would like to include in our framework dedicated fragments and mechanisms to support traceability through the analysis and design stages of the development process.


## References


- [1] Jacobson I. Booch G., Rumbaugh J. *The Unified Modeling Language User Guide*. Object Technology Series. Addison-Wesley, 1998.
- [2] E.F. Codd. Further normalization of the data base relational model. In '*Data Base Systems*', Rustin(ed), Prentice-Hall publishers. 1972.
- [3] A. Flory and J. Kouloumdjian. A model and a method for logical data base design. In S. Bing Yao, editor, *Very large data bases: Fourth International Conference on Very Large Data Bases, West Berlin, Germany, September 13–15, 1978*, pages 333–341, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1978. IEEE Computer Society Press.
- [4] Object Management Group. The UML notation. <http://www.omg.org/>.
- [5] I. Mirbel and V. de Rivieres. Towards a UML profile for building on top of an existing application. In *Information Resources Management Association International Conference*, Seattle, USA, May 2002.
- [6] J. Ralyte. *Ingenierie des methodes a base de composants*. PhD thesis, Universite Paris I - Sorbonne, January 2001.
- [7] J. Ralyte and C. Rolland. An assembly process model for method engineering. In M.C. Norrie K.R. Dittrich, A. Geppert, editor, *Advanced Information Systems Engineering*, number 2068 in LNCS, pages 267–283. Springer, June 2001.
- [8] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Object Technology Series. Addison-Wesley, 1998.
- [9] K. van Slooten and B. Hodes. Characterizing is development projects. In R.J. Welke S. Brinkkemper, K. Lyytinen, editor, *IFIP TC8, WG 8.1/8.2*, pages 29–44, August 1996.
- [10] I. Vessey and R.L. Glass. Applications-based methodologies. *Information System Management*, pages 53–57, Fall 1994.
- [11] M. Zamfiroiu and N. Prat. *Ingenierie des systemes d'information*, chapter 9. Herms, 2001.


## 6. Appendix

REQUIREMENTANALYSIS::CORE::INTERLEAVING	
<b>Situation</b>	<input checked="" type="checkbox"/> /Functionalities or Interface preservation
<b>Associated Fragments</b>	Current::Refinement::Interleaving
<b>Intention</b>	Use-case
<b>Guidelines</b>	<p>UML Diagrams:</p> <ul style="list-style-type: none"> <li>▷ To distinguish the new parts from the existing ones, as well as the existing parts taken as they are and the ones to be modified.</li> </ul> <ul style="list-style-type: none"> <li>▷ If <i>functionalities</i> or <i>code</i> have to be preserved, in the use-cases resulting from the <i>Basic Fragment</i> use stereotypes to distinguish new functionalities (&lt;&lt;new&gt;&gt;), existent and modified functionalities (&lt;&lt;to-be-modified&gt;&gt;) and preserved functionalities (&lt;&lt;re-use&gt;&gt;).</li> <li>▷ If <i>interfaces</i> have to be 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 some constraints which have to be identified through the requirement analysis phase. Most of all, systems already collaborating with the running part of the application have to be identified, because they may use an interface that therefore can't be changed. Indeed, these actors represent a strong constraint for the new development. The proposed stereotypes help in distinguishing systems already collaborating from the other ones. <ul style="list-style-type: none"> <li>▷ <b>System:</b> an actor interacting with the application and being a system (and not a person).</li> <li>▷ <b>Human:</b> a person using the application (through a man-machine interface).</li> <li>▷ <b>Dependent system:</b> a system already interacting with the running application and willing to continue its collaboration.</li> <li>▷ <b>New system:</b> a system which will use the services of the application under development.</li> <li>▷ <b>Constraining system:</b> 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 can't be changed at all. The actor imposes constraints to the application.</li> <li>▷ <b>Collaborating system:</b> a system already interacting with the application. The actor will continue to interact with the application but its interaction mode may be slightly modified.</li> </ul> </li> </ul>

REQUIREMENTANALYSIS::REFINEMENT::INTERLEAVING	
<b>Situation</b>	<input checked="" type="checkbox"/> /Functionalities or code preservation
<b>Associated Fragments</b>	Current::Core::Interleaving, Current::Current::Processing-Order
<b>Intention</b>	<ul style="list-style-type: none"> <li>▷ To highlight new functionalities inside use-cases taken from the running application. A split may also lead to the discovery of &lt;&lt;out-of-scope&gt;&gt; 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 &lt;&lt;re-use&gt;&gt; and &lt;&lt;out-of-scope&gt;&gt; stereotypes.</li> </ul>
<b>Guidelines</b>	<p>UML Diagrams:</p> <ul style="list-style-type: none"> <li>▷ When a new functionality is introduced in an use-case corresponding to a running functionality(r-uc), represent it as a different use-case related to the use-case r-uc with an inclusion relationship.</li> </ul>

REQUIREMENTANALYSIS::INTRODUCTION::SUBSYSTEM FRAGMENT	
<b>Situation</b>	 /
<b>Associated Fragments</b>	
<b>Intention</b>	<ul style="list-style-type: none"> <li>▷ To give a general view of the software and to show how it is distributed.</li> </ul>
<b>Guidelines</b>	UML Diagrams: Activity
	<ul style="list-style-type: none"> <li>▷ Use activity diagrams with the notion of swim lane to distinguish the different subsystems of the software. Tasks under the responsibility of an activity are represented by an activity in the column corresponding to the subsystem.</li> <li>▷ Through the main step, use case diagrams will be given for each subsystem separately. Therefore, each subsystem will appear as an actor in the use case diagrams describing the functionalities of the other subsystems it is related to.</li> <li>▷ The diagram should include the processing related to the normal flow but also to the exceptional situations.</li> </ul>

REQUIREMENTANALYSIS::CORE::SCREEN-SHOT	
<b>Situation</b>	 /
<b>Associated Fragments</b>	Current::Refinement::Screen-Shot, Current::Current::Screen-Shot-Sequence, Domain&BusinessOnjectAnalysis::Refinement::Screen-Shot
<b>Intention</b>	<ul style="list-style-type: none"> <li>▷ To complete use-case description with screen shots.</li> </ul>
<b>Guidelines</b>	UML Diagrams: Use-case
	<ul style="list-style-type: none"> <li>▷ For each use-case related to a human actor, provide screen shots in a dedicated section.</li> <li>▷ Information about labels, input and output data, icons and so on have also to be detailed in the screen-shot section.</li> <li>▷ Distinguish use-case including a GUI by using a dedicated stereotype, for instance, &lt;&lt;user-related&gt;&gt;.</li> </ul>

REQUIREMENTANALYSIS::CORE::PERSISTENCE	
<b>Situation</b>	 /
<b>Associated Fragments</b>	
<b>Intention</b>	<ul style="list-style-type: none"> <li>▷ To highlight use-cases accessing the database.</li> </ul>
<b>Guidelines</b>	UML Diagrams: Use-Case
	<ul style="list-style-type: none"> <li>▷ Distinguish use-case related to the database by using a dedicated stereotype: &lt;&lt;persistent&gt;&gt;.</li> </ul>