

# Intelligent environment for rapid process design

Bilal Hussein<sup>1</sup>, Patrice Caulier<sup>1</sup>, Moncef Yousef<sup>2</sup>, Patrik Millot<sup>1</sup>, Yahia Rabih<sup>3</sup>,

<sup>1</sup> University of Valenciennes and the Hainaut-Cambrésis - Laboratory of automatic, of Mechanics and industrial and Human data processing LAMIH UMR CNRS 8530 Campus du Mont Houy – F-59313 VALENCIENNES CEDEX 9- FRANCE

husseinbilal@yahoo.fr

patrice.caulier@univ-valenciennes.fr

millot@univ-valenciennes.fr

<sup>2</sup> Lebanese University – engineering Faculty - Branch 2 - Fanar - LEBANON

ymonsef@ul.edu.lb

<sup>3</sup> Lebanese university - Beirut - LEBANON

yrabih@hotmail.com

---

*Abstract: The domain of this article is the software engineering for rapid process design. It aims to define a new methodology of software development based on an intelligent mixed environment (IME) consists in knowledge base and case based reasoning. The IME allows a designer/analyst to generate automatically a design model by using a set of standard requirements defined as scenarios.*

*The article presents the stages of IME development. It explains the role of job experts during the process of requirement standardization and exposes a new philosophy of software life cycle based on the micro-requirement concept. This role consists in transforming the expert informal knowledge (micro-requirements) in formal knowledge (scenarios of use cases) and regrouping them in a knowledge base. Then, the system engineer and the customer use the IME to build the requirements model (use cases scenarios) with probably introducing some parameters. Finally, The system engineer generates, through the scenarios classes, a design model in UML formalism using CBR techniques.*

*KEY WORDS: Case based Reasoning (CBR), knowledge base, software engineering, use case, requirement engineering, software life cycle, micro-requirement.*

---

## 1. Introduction

This article appears in the domain of intelligent software engineering. It aims to define a new methodology of software development based on an intelligent mixed environment (IME) which consists in knowledge base and case based reasoning (CBR). The IME allows a designer/analyst to generate automatically a design model using a set of standard requirements predefined as scenarios.

The problematic of software engineering, is due to the presence of several methods of design and development and their different interpretations [1]. In addition to this, a bad understanding of the users requirements has a direct influence on the success of a computer project.

In 1999 the results, of a study made on the Department of Defense Software (DoD), published in the conference " 5th Annual Joint Aerospace Weapons Systems Support, Sensors, and Simulation Symposium ", showed that only 2% of the computer projects achieved by (DoD) have been used and that 75% of these projects have been rejected or never used. The 23% remaining projects have been used after modification [2]. Another study achieved by Standish Group in 1994 on 8000 projects led by 350 companies, produced similar results. Only 16% of the projects have been considered successful [3].

Leishman and Cook [2] send back this to the presence of an unsuitability between the services provided by the system after its realization and the user requirements that, normally, are fixed in the beginning of the development process. This unsuitability is due to the absence of a common language between the customer and the system engineer (expert in analysis and design).

"Intelligent software engineering" will propose to develop, using new software life cycle approach and new technique of requirement engineering, a system aided design able to produce rapidly a design model. Intelligent software engineering process evolves in a knowledge base environment and use intelligent design tools, allowing system engineers to build models using standard requirements.

"Intelligent software engineering" aims to:

- Produce rapid specification models for unexpected modifications on requirements (example real time systems).
- Use the techniques and the capacities of artificial intelligence in the software engineering process.
- Design user's needs as knowledge rather than data.

## 2. System life cycle and requirement engineering

In the information system life cycle we can distinguish three big periods: design, realization and maintenance [4]. The period of the maintenance consists in the evolutions appearing after the operational launching and in the modifications of the initial application to adapt it to unforeseen changes in the requirements [4].

Unfortunately, the reality suggests that an excessive percentage of software development resources is spent for the preservation of programs (maintenance) [5].

The present methods of information system design don't permit to conceive artificial information systems endowed with capacities of auto-adaptation and auto-evolution and that permit them to come with the evolution of the enterprise and its natural information system [4].

To decrease the cost of the maintenance the research has been focused on introducing new approaches in system life cycle, and improving the techniques and methods of requirement engineering.

### 2.1. Reuse approaches in software life cycle

Over the past 20 years, many techniques have been developed to support software reuse. These exploit the facts that systems in the same application domain are similar and have potential for reuse. That reuse is possible at different levels (from simple function to complete application, and that standards reuse components facilitate reuse. The following (figure.1) illustrates the number of ways to support software reuse [6].

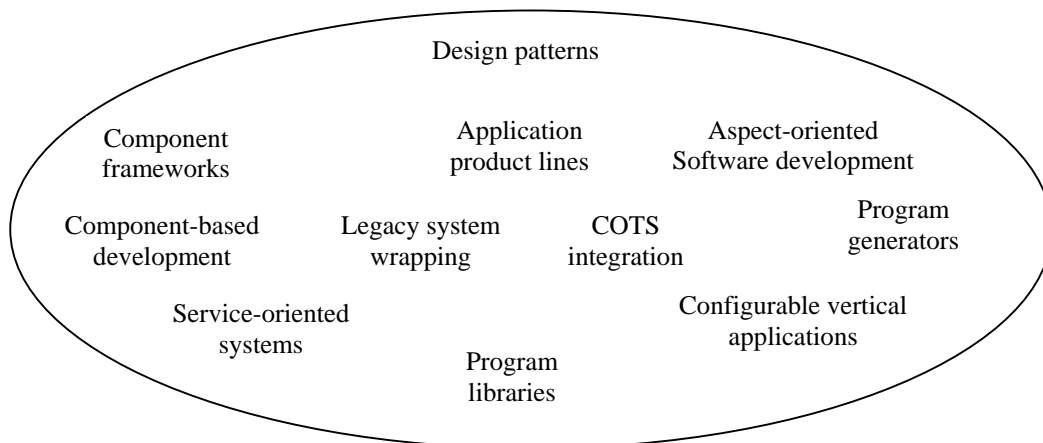


Figure 1. Reuse techniques landscape

#### 2.1.1. Design pattern approach

The pattern is a description of the problem and the essence of its solution, so that the solution may be reused in different settings [6].

Patterns propose different families of structures that are frequently encountered and that user is invited to try on the treated problem to benefit of them. So, patterns can be seen like elaborate way to reuse knowledge acquires by experience [7].

The use of patterns is an effective form of reuse, but only experienced software engineer who have a deep knowledge of patterns can use them effectively [6].

The choice of a suitable pattern then adapt it to a new problem is a very hard work for a software engineer because, firstly, he should understand in detail user requirements.

### 2.1.2. Generator-based reuse approach

The Generator-based reuse approach has been founded by Biggerstaff (in 1998). In this approach, reusable knowledge is captured in a program generator system that can be programmed by domain experts. The application description specifies, in an abstract way, which reusable components are to be used, how they are to be combined and their parameterization. Using this information, an operational software system can be generated [6].

Generator-based reuse takes of the fact that applications in the same domains (such as business systems) have common architectures and carry out comparable functions (Figure 2).

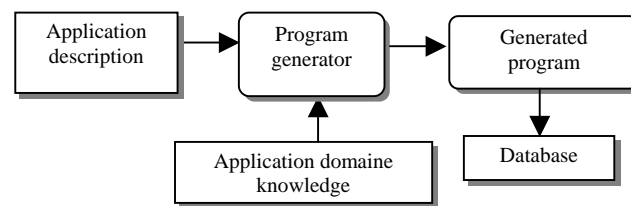


Figure 2. Generator-based reuse

### 2.1.3. Component-based software engineering (CBSE) approach

Component-based software engineering (CBSE) is a process that emphasizes the design and construction of computer-based system using reusable software “components” [8]. Designer and programmer build their system by integration of components. These components must possess communication interfaces to exchange information.

Component-based development is a minor phase in relation to software life cycle. The assembly is a simple process. It is sufficient to select the good components, for a good application, and integrate them in the same system.

Currently, the research aims to create a compatibility protocol between the different components to assure their interoperability.

But, What can we do when executable component doesn't answer our requirements exactly? What can we do when the component source code is not accessible by designers and developers?

In fact, using reuse approaches in software development provides many advantages in cost and time. However, it doesn't success only if system engineer has deeply understood user requirements.

## 2.2. Problematic of requirement engineering

The requirement engineering is one of the methods permitting to minimize the risk of unsuitability of the product or service to the customer's needs [9].

The important research on requirement engineering is achieved by RENOIR group (International Group work on requirement engineering). According to RENOIR, the requirement engineering tries to carry some improvements to life cycle level of system development. It provides tools, concepts and methods that put in relation supplier services and information technology products [9].

The requirement engineering proposes process scenarios to identify and to collect requirements. These processes have been criticized by a work done in 1992 titled “Issues in requirements elicitation » [10]. This work focuses on the problematic of requirement identification and proposes a conceptual methodology setting combining the advantages of different engineering methods [9]. This methodology setting proposes a process of requirements compilation in five steps:

- List of facts.
- Requirements collection and classification.

- Assessment and rationalization.
- Hierarchization.
- Integration and validation.

Several techniques and methods have been developed according to this methodology setting like PREVIEW, NATURE, MAP, CREWS, CREWS-l'Ecritoire.

All these methods used the process of requirement compilation in different ways because each method rests on a concept different of the other.

PREVIEW uses the concept of viewpoint; NATURE uses the concept « Situation, Decision, Argument, Action »; MAP is an extension of NATURE project uses a driven model of several engineering models; CREWS uses the concept of scenario; and CREWS-l'Ecritoire uses a concept that couples between goals and scenarios.

So, the success or the failure of one of the above methods depends on several criterions:

- Selection of relevant model.
- Understanding requirements by system engineer.
- Understanding the relevant model.
- Complexity of the system.

### 3. New development approach based on IME

In this section we present our new approach of software development and design methodology. This approach is based on intelligent mixed environment (IME) consists in knowledge base and case based reasoning. The IME allows a system engineer to generate, in automatic way, a design model by using a set of standard requirements predefined as scenarios.

#### 3.1. Philosophy of the life cycle

Whatever is the life cycle approach, there is always something remaining in question: How was the beginning of the life cycle?

The software life cycle is very important in the philosophy of software engineering. But, the question asked is: " Are the elements created at the root (beginning of the life cycle) correct?"

In genetic science, the man is hit of an illness after 50 years because his gene, since the birth, is badly formed.

Can we create, during the process of baby's formation, correct genes? The answer is today in the genetic engineering research laboratories. But, with software engineering is a little difference because man builds the software genes. So, the creation of correct software genes is possible.

#### 3.2. New approach and methodology

Let's suppose that a system is composed of a set of components containing each one a set of cells where each one is formed of a set of software genes *sg* (Figure. 3).

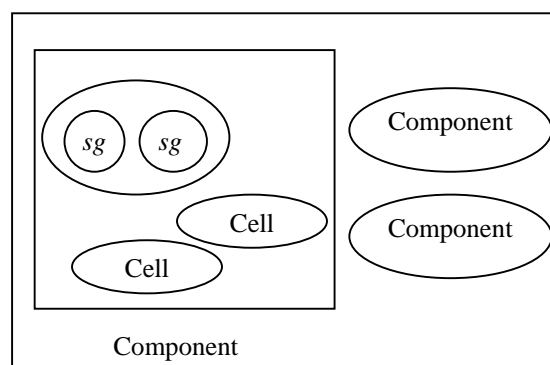


Figure 3. Software genes based system

A gene badly formed can cause a bad working of the cell, of its component and therefore of the whole system. Thus, the whole system life cycle is affected. So, it is necessary to create, since the beginning, correct genes that can enlarge system life cycle.

The cell (component object) is the composition of several sg. The sg represents a micro-requirement that can be an object property or an object operation. Therefore, The construction of a correct cell requires a standardization of the requirements.

Can we standardize requirements? Can we say that a requirement specification is universal?

François Coallier assures that all elements constituting the software life cycle can be standardized [11]. He presented the different norms of standardization and the future axis of the software engineering standardization under the SC7 norm.

### 3.3. General process for IME development

In this paragraph we are going to present the stages to develop the IME (Figure 4).

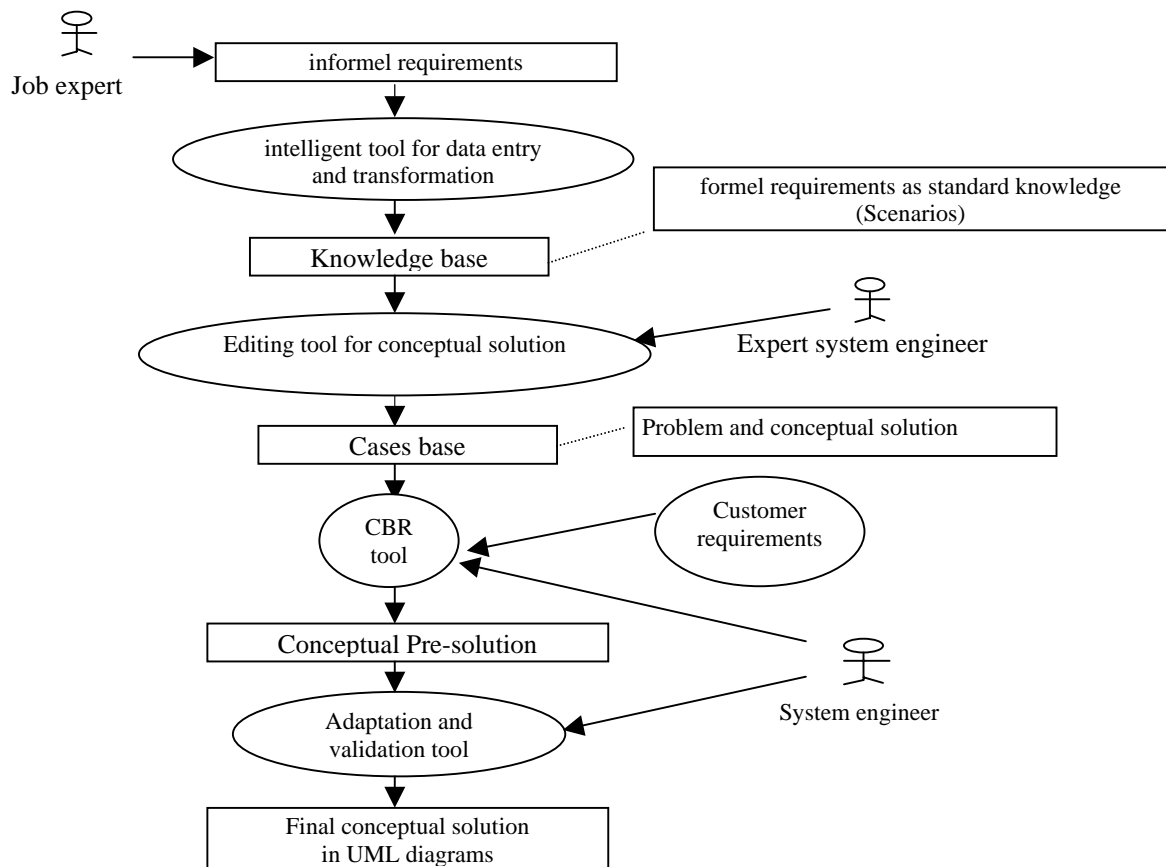


Figure 4. development process of IME

The IME permits to enter the knowledge of a job expert as scenarios and allows the system engineer and the customer to identify their functional requirements with the possibility of parameterization. The concept of scenario illustrates the dynamic problem solving and it is used in different application domains with different senses [15].

The stages of this process are the following:

- Establish a data entry and transformation tool to capitalize the informal requirements of the job expert as formal standardized knowledge (Prototype Scenarios) [12]. This tool must allow the job expert to:
  - Decompose his knowledge as fragments of requirements (micro-requirements).
  - Design these fragments of requirements by fragments of prototype scenarios.
  - Compose the resulting fragment scenarios to build prototype scenarios and regroup them in use cases classes. Each use case class is saved in a knowledge base. This

knowledge base is enriched by inference rules to develop new scenarios by combination of requirement fragments.

- Establish an editing tool for conceptual solutions in order to complete the basis of CBR system. To make the knowledge base similar to a base used in CBR system, the system engineer adds the conceptual solution to each scenario existing in the use case classes.
- Generate, using a CBR tool, a conceptual pre-solution corresponding to the customer requirements.
- Adapt and validate the pre-solution, in presence of the customer, in order to get the final conceptual solution in UML diagrams.

### 3.3.1. CBRs and micro-requirements knowledge

The case based reasoning (CBR) is a paradigm of problems resolution that tries to solve a target problem while leaning on a past resolved base cases [13].

Other definition, CBR is used to solve a new problem by remembering a previous similar situation and by reusing information and knowledge of that situation [14].

The first two tools in IME are used to build the cases base in which domain requirements and their design models are integrated (Figure 5.).

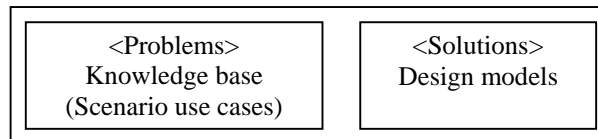


Figure 5. Cases base

#### a) Knowledge base

**Use case illustration example:** Withdraw money through an automated teller machine (ATM). Several standard prototypes scenario belong to different ATM agencies.

The nominal scenario of ATM1 machine:

- Customer inserts credit card.
- ATM1 validates the card.
- Customer enters password.
- ATM1 checks password.
- Customer chooses the option fast withdraw.
- ATM1 displays two currencies: Lebanese pound and American Dollar.
- Customer chooses the currency Dollar.
- ATM1 displays several possibilities of amounts: 50, 100, 200, 300, 400, 600,.
- Customer chooses an amount.
- ATM1 validates the amount, delivers the money and prints a receipt.
- Customer takes card and money.
- ATM1 is reinitialized and ready for another operation.

The nominal scenario of ATM2 machine:

- Customer inserts credit card.
- ATM1 validates the card.
- Customer enters password.
- ATM2 checks password.
- Customer chooses the option fast withdraw.
- ATM2 displays two currencies: Lebanese pound and American Dollar.
- Customer chooses the currency Dollar.
- ATM2 displays several possibilities of amounts: 20, 40, 60, 100, 200, 300,.
- Customer chooses an amount.
- ATM2 validates the amount, delivers the money and prints a receipt.
- Customer takes card and money.
- ATM2 is reinitialized and ready for another operation.

The nominal scenario of ATM3 machine:

- Customer inserts credit card.
- ATM1 validates the card.
- Customer enters password.
- ATM3 checks password.
- Customer chooses the option fast withdraw.
- ATM3 displays several possibilities of amounts in Euro: 20,50, 100, 200, 300, 400,.
- Customer chooses an amount.
- ATM3 validates the amount and delivers the money.
- Customer takes card and money.
- ATM3 is reinitialized and ready for another operation

The three above scenarios share some operations and distinguish in others. Each use case scenario can be described by the following operations:

- Trigger operations.
- Main operations.
- Secondary operations.
- Stop operations.
- Specialize operations.

Trigger operations: Insert card and validate card.

Main operations: enter password, check password, select amount and authorize amount.

Secondary operations: Control moneybox and eject money.

Stop operations: Reinitialize, take card and take money.

Specialize operation for ATM1 machine:

- ATM1 displays two currencies: Lebanese Pound and American Dollar.
- Customer chooses the currency Dollar.
- ATM1 displays several possibilities of amounts: 50, 100, 200, 300, 400, 600,.
- ATM1 prints a receipt.

Specialize operation for ATM2 machine:

- ATM2 displays two currencies: Lebanese Pound and American Dollar.
- Customer chooses the currency Dollar.
- ATM2 displays several possibilities of amounts: 20, 40, 60, 100, 200, 300,.
- ATM2 prints a receipt.

Specialize operation for ATM3 machine:

- ATM3 displays several possibilities of amounts in Euro: 20,50, 100, 200, 300, 400,.

The different specialize operations are defined in different subclasses points of views (Figure 6).

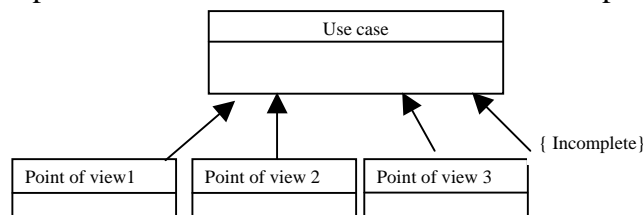


Figure 6. Standard prototype

The trigger, main, secondary, stop and specialize operations represent the fragment scenarios that are already inputted by the job expert (Figure 7).

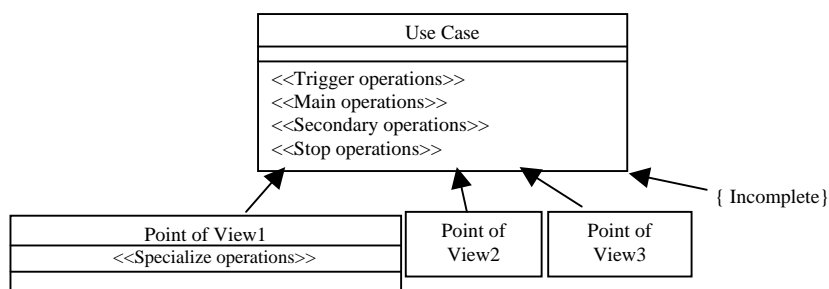
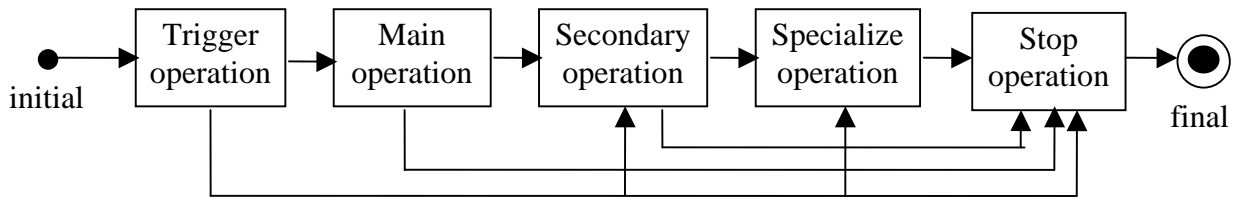


Figure 7. Detail of a prototype use case

How to build knowledge base?

According to steps indicated in paragraph 3.3, the proposed tool will help job expert to build knowledge base as set of scenarios. Expert should enter his scenarios like chain of abstract operations without any detail. The knowledge syntax will be:



The operation format will be:

<Object name><Operation category>,<level>,<Abstract operation name>,<[input;..]>,<[output;..]>

example:

- ATM,Trigger,1,InsertCard(),[boolean]
- ATM,Trigger,2,Validate\_card (), [card number, expiry date], [boolean]
- ATM,Main,1,Enter\_password (), [password]
- ATM,Main,2,Check\_password (),[password],[boolean]
- ATM,Main,3,Select\_amount (),[amount]
- ATM,Main,4,Authorize (),[card number, amount],[sold]
- ATM,Secondary,1,Control\_box (), [amount], [boolean]
- ATM,Secondary,2,Eject\_money (),[amount], [boolean]
- ATM,Stop,1,Take\_Money (),[boolean]
- ATM,Stop,2,Take\_Card (), [boolean]
- ATM,Stop,3,Reinit()
- ATM,Specialize,1.1, Display\_amount\_\$ () /\* first subclass
- ATM,Specialize,1.1, Display\_amount\_LL () /\* first subclass
- ATM,Specialize,1.1, Select\_currency (), [currency] /\* first subclass
- ATM,Specialize,1.1, Print\_receipt (), [amount], /\* first subclass
- ATM,Specialize,2.1, Display\_amount\_\$ () /\* second subclass
- ATM,Specialize,2.1, Display\_amount\_LL () /\* second subclass
- ATM,Specialize,2.1, Select\_currency (), [currency] /\* second subclass
- ATM,Specialize,2.1, Print\_receipt (), [amount], /\* second subclass
- ATM,Specialize,3.1, Display\_amount\_euro () /\* third subclass

The first *IME* tool builds through the chain of abstract operations the use case classes depend on operation categories (Figure 8).

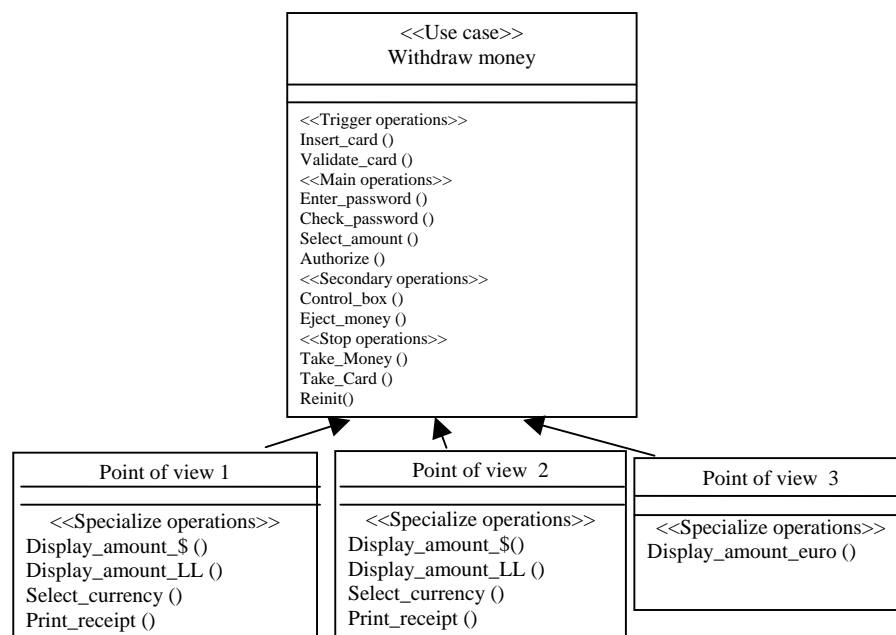


Figure 8. ATM Prototype

b) Design model

The static aspect of requirements (scenarios) will be represented by attributes in use cases class (Figures 10,11).

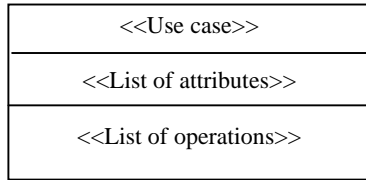


Figure 10. General use case

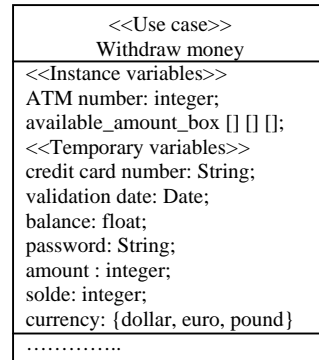


Figure 11. Use case – static aspect

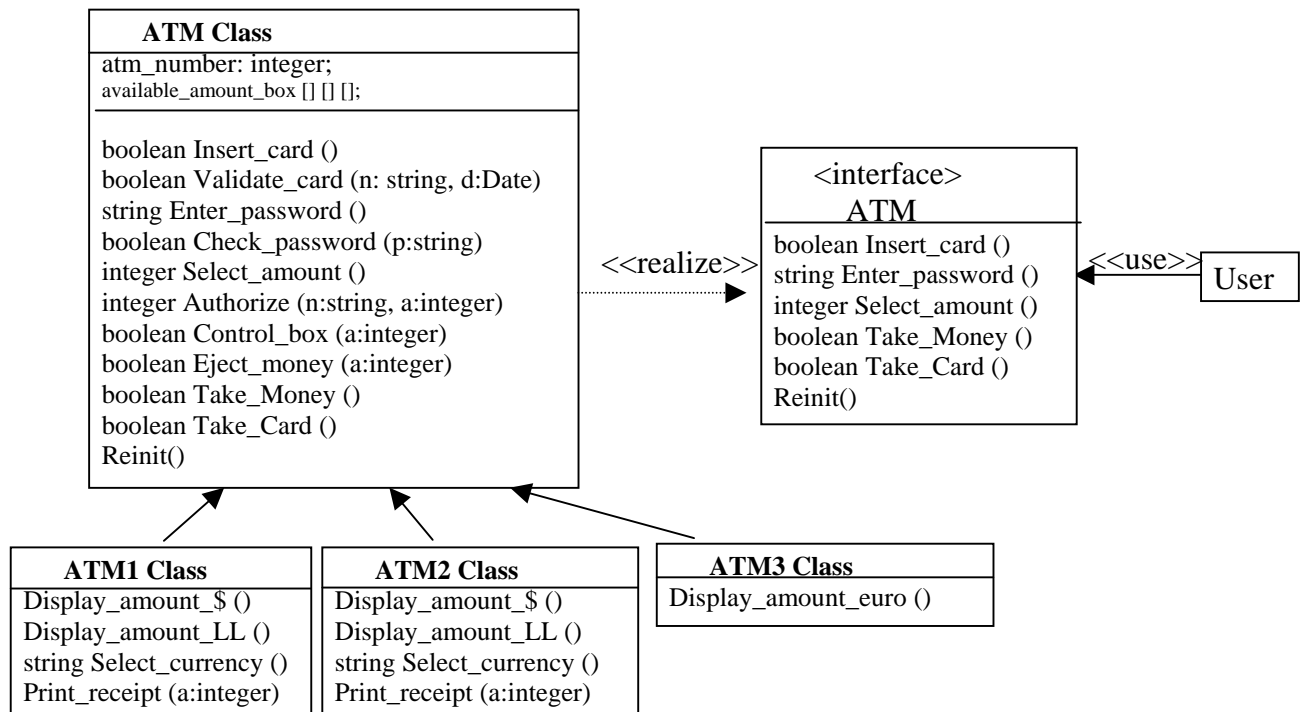


Figure 12. Design model

3.3.2 Problem specification and parameterization of standard scenarios

The first step of problem resolution using a CBR is the problem classification and specification [13]. Certainly, this step is the most important in CBR cycle because the success of other steps depends on its success.

In our approach we consider that the customer requirements and his expectations represent the problem to solve. Therefore, The system engineer, must firstly, builds the specification of the problem to solve in order to elaborate its solution.

To carry through this objective, the system engineer proposes to the customer, relatively to the problem class, several requirement prototypes (use case classes) in order to allow him the identification of the set of requirements with the possibility of modification (Figure 13).

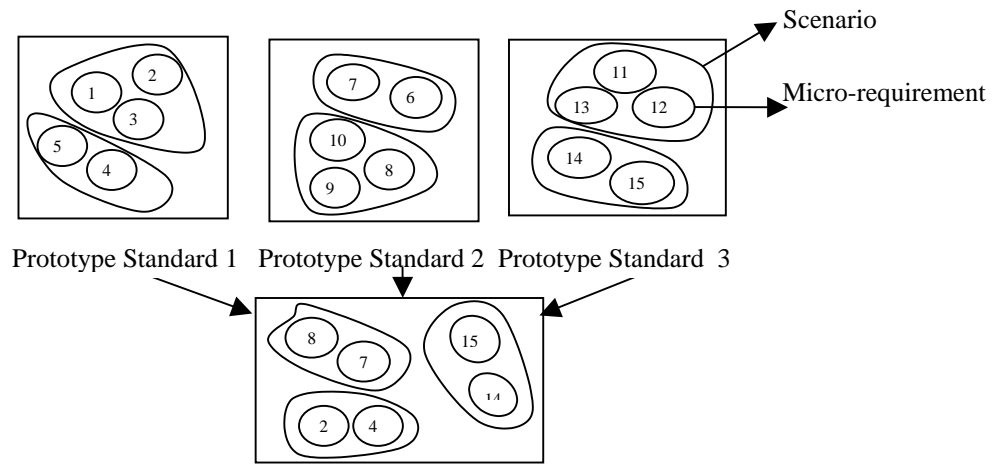


Figure 13. Formal requirement obtained through three standard prototypes

At the end of this step a use case model will represent the specification of the target problem.

The customer composes his scenarios through available standard scenarios, and then he can add to them parameters in order to introduce private cases.

These parameters are represented by specialization operations that are specific to customer and forming his point of view (Figure 14).

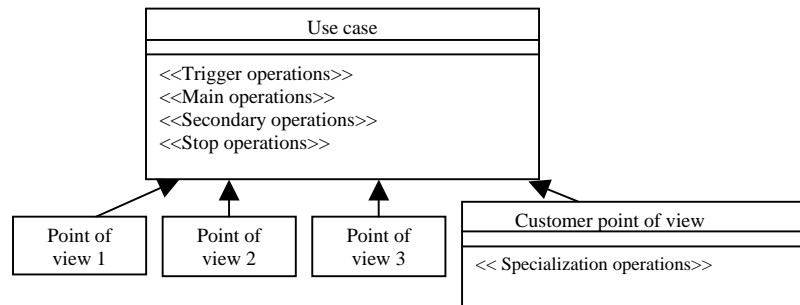


Figure 14. Prototype modified using customer point of view

A use case scenario represents the scenario that corresponds to a set of micro-requirements or a set of software genes sg. The system engineer must collect all sg then builds a use case model that expresses the customer's requirements.

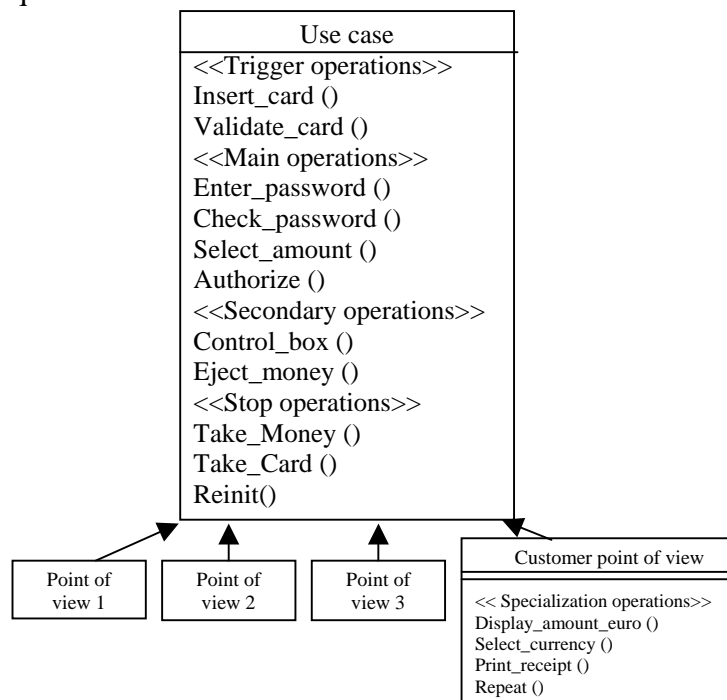


Figure 15. Prototype modified using customer point of view

### 3.3.3 generation of the design model

#### *a) Search a similar case and its solution*

The second step of problem resolution using CBR is to find, in the cases base, a case similar to the constructed specification and to exploit its solution.

To determine the similarity between two cases is far from being a trivial operation. It implies the use of measures and knowledge of greatly linked similarities in the application domain [13]. Some generic measures of similarity have been proposed, and a survey of these is proposed in (Rifqi, 1996).

The technique that we use to build the target use case model can offer advantages to the problems of similarity because during the specification phase the system engineer and the customer have used, at certain point, prototype scenarios that are saved in the cases base. Therefore, an important part of the solution is already available. It remains to find, in the cases base, the cases similar to the parameterized scenarios that represent customer's point of views.

When the source case is identified completely, the system engineer, using a CBR tool, builds, firstly, the pre-solution while bringing the parameterized scenarios of the memorized scenarios closer possible. This, it is made by the decomposition of the parameterized scenarios in fragments of scenarios in order to facilitate the test of the scenario in relation to others.

This technique permits to rebuild the conceptual solution, corresponding to the scenario, step-by-step. The technique will be iterative and it will be applied on the whole use case model in order to construct the complete pre-solution.

#### *b) Adaptation and validation of pre-solution*

The third step of problem resolution using CBR is called adaptation. There are two major steps involved in adaptation: figuring out what needs to be adapted and doing the adaptation [16].

The pre-solution should be adapted then validated by using a tool of adaptation and validation. The results of this step should be answer exactly to customer's needs. The adapted and validated pre-solution becomes a final solution that represents the design model of use case model.

## **4. Conclusion**

The standardization of requirements is a very important phenomenon in the software life cycle. It permits, probably, to fill the ditch between the user and the system engineer that don't get along at the time of development cycle. The standardization of requirements reduces the development cost and open the door owing a new vision of industrialization.

Therefore, we are bringing to use and to exploit knowledge and experiences of experts in all domains. For never loose some successes experiences, it is necessary to resort to the CBR systems and to enrich them by new knowledge and experiences.

Job experts are called to participate deeply in the development of knowledge bases while hoping that we will be able to assure for the humanity the ability of transformation toward digital societies where all operation or phenomenon of the society will be computerized.

## 5. References

- [1]. Michael Bowman, Antonio Lopez, James Donlon, Gherghe Tecucci, Teaching Intelligent agents: Software design methodology, CrossTalk: The journal of Defense Software Engineering June 2001.
- [2]. Theron R. Leishman, Dr David A. Cook - Requirements Risks Can Drown Software Projects – CrossTalk: The Journal of Defense Software Engineering Apr 2002.
- [3]. The Standish Group International, Inc. The CHAOS report, 1994.
- [4] Dominique Nancy, Bernard Espinasse - Ingénierie des systèmes d’informations, MERISE-Cybex troisième édition 1996.
- [5] Grady Booch, Analyse et conception orientées objet - Addison Wesley. 2me édition 1994.
- [6]. Ian Sommerville, Software Engineering Seventh Edition - Addison Wesley 2004
- [7] Christine Choppy, Maritta Heisel, Une approche à base de “patrons” pour la spécification et le développement de systèmes d’information, AFADL, Besancon, France, June 2004.
- [8] - Roger S. Pressman Software Engineering, Sixth Edition: A practitioner's Approach - McGraw-Hill. 2004
- [9] Jérôme Costanzo, Méthodes pour l’ingénierie du besoin complexe - mémoire DEA 2000.
- [10] Christel, M.G. Kang and, K.C, Issues in Requirements Elicitation, Technical report. CMU/SEI-92-TR-12, September 1992.
- [11]. François Coallier - International Standardization in Software and Systems Engineering. Journal of Defense Software Engineering 2003.
- [12] Valentina Ceausu, Sylvie Despres, Utilisation de ressources sémantiques pour l’élaboration et la remémoration de cas, 13ème Atelier Ràpc 2005.
- [13] Amélie Cordier, Béatrice Fuchs, Un assistant pour la conception et le développement des systèmes de Ràpc, 13ème Atelier Ràpc 2005.
- [14] Agnar Aamodt, Enric Plaza Case-based reasoning: Foundational Issues, Methodological Variations, and System Approaches, AICom Vol.7 No.1 Mars 1994.
- [15] Mejri Lassaâd, Caulier Patrice, Formalization of a scenario concept to dynamic problem solving, EMA’2005 Athenes- October 2005.
- [16] Janet L. Kolodner, An introduction to case-based reasoning, Artificial Intelligence Review V6, 1992.