L'E-Lyee, Coupling L'Ecritoire and LyeeALL

Colette Rolland

L'E-Lyee: Coupling L'Ecritoire and LyeeALL

C. Rolland
Université Paris1 Panthéon Sorbonne
CRI, 90 Rue de Tolbiac
75013 Paris, France
rolland@univ-paris1.fr

Abstract

The paper deals with the requirements engineering environment provided by L'Ecritoire to the L'E-Lyee project. The project aims to reduce the software development cycle to two explicit steps, requirements engineering and program generation, by coupling L'Ecritoire to the program generation features of LyeeALL. The basis of L'Ecritoire is a set of enactable rules to guide the requirements elicitation process through interleaved goal modelling and scenario authoring. The paper gives an overview of the enactment rules and illustrates their use through a L'Ecritoire session. Thereafter, the matching of the technical features of L'Ecritoire with those of LyeeALL is outlined and the resulting benefits are highlighted.

1. Introduction

L'E-Lyee is a CASE environment which supports software development in 2-steps, requirements engineering and code generation. The former is provided by L'Ecritoire and the latter by LyeeALL.

LyeeALL is a commercial Japanese CASE environment. The right hand side of Figure 1 shows that the underlying Lyee approach comprises a framework to structure programs and control their execution, and a generation mechanism to generate programs from given software requirements. These requirements are expressed in rather low-level terms such as screen inputs and outputs, formulae for output calculation and data base accesses. LyeeALL has been used in a number of large companies such as Mitsubishi. This experience shows the need to acquire software requirements in a systematic way from high level system requirements.

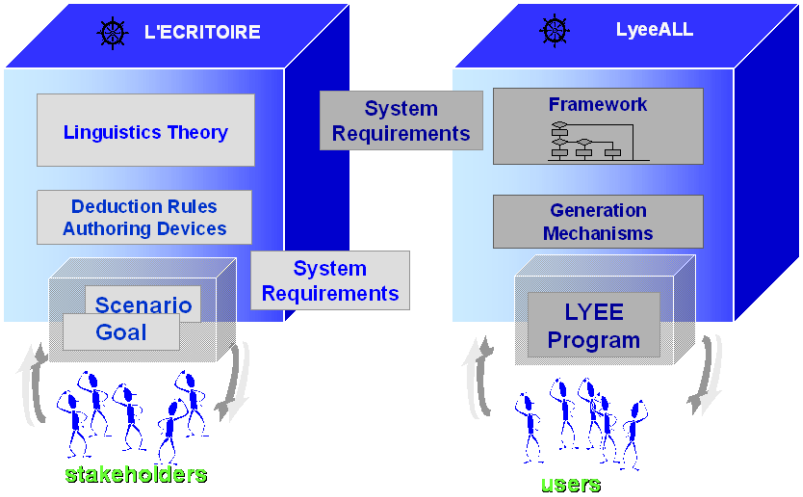

Figure 1: The L'E_Lyee Project

The L'E-Lyee project[1] is a Franco-Japanese effort towards meeting this need. The project aims to couple together the French research prototype called L'Ecritoire, a tool for requirements engineering with LyeeALL thereby addressing the entire system development life cycle. The left hand side of Fig. 1 shows that the approach underlying L'Ecritoire uses goal-scenario coupling to discover requirements from a computer-supported analysis of textual scenarios. L'Ecritoire produces a requirements document which relates software system requirements to organisational goals. The aim of the L'E_Lyee project is to match the L'Ecritoire software system requirements to the LyeeALL software requirements.

In this paper we concentrate on L'Ecritoire and present a usage scenario to illustrate its functionality. The matching of the technical features of L'Ecritoire with those of LyeeALL is outlined. However, the details of this matching are the subject of another paper.

This paper is organised in two main sections. The first presents an overview of the L'Ecritoire approach: the goal-scenario coupling, the notion of a requirement chunk, scenario authoring, goal discovery and associated enactable rules. In the third section, the functionality of L'Ecritoire is illustrated through a usage scenario. The concluding section contains a discussion of the L'Ecritoire-LyeeALL coupling. It also identifies the benefits of this coupling.

2. Overview of the L'Ecritoire Approach

L'Ecritoire is a tool for requirements elicitation, analysis and verification, structuration, and documentation. It draws heavily from the work on goal, scenario, and goal-scenario coupling found in the literature. *Goal-oriented requirements engineering* and *scenario-based requirements engineering* are two distinct trends aiming at eliciting requirements from an analysis of the wider context in which the system will operate. A *scenario* is 'a possible behaviour limited to a set of purposeful interactions taking place among several agents' [4,11]. It describes a desirable functionality of a system under design, and thus, helps in identifying requirements [12]. The problem with scenarios is that they are inherently partial and therefore they raise a coverage problem making it impossible to verify the completeness of the requirements[5].

Goal modelling is another way to facilitate requirements elicitation. Even though goal modelling has been found to be highly appropriate [10], experience shows that it is difficult for domain experts to deal with the fuzzy concept of a goal [3, 1]. Though it is assumed that systems are constructed with some goals in mind [7], experience [1, 17] shows that goals are not given and therefore the question of where they originate from [1] acquires importance. In addition, the goals often given by organisations are not real ones but reflect an idealised view of the enterprise. Therefore, proceeding from these may lead to ineffective requirements [13]. Thus, goal discovery is rarely an easy task.

The goal-scenario combination has been used to operationalise goals [5]. This is because scenarios can be interpreted as containing information on how goals can be achieved. This suggests a unidirectional relationship between goals and scenarios. In L'Ecritoire, however, we view *this relationship as bi-directional* : just as goals can help in scenario discovery, so also scenarios can help in goal discovery. The total solution is in two parts. First, for a goal,

[1] This work is supported by the Institute of Computer Software methodology and Technology, Tokyo, Japan, under a Joint Research Agreement between UP1 and the Institute.

scenarios are authored. Thereafter, the authored scenario is explored to yield goals which, in turn, cause new scenarios to be authored and so on.

2.1 The notion of a requirement chunk

At the core of the L'Ecritoire approach is the notion of a *Requirement Chunk*. We define a *Requirement Chunk* (RC) as a pair <G, Sc> where G is a goal and Sc is a scenario. Since a goal is intentional and a scenario is operational in nature, a requirement chunk is a possible way in which the goal can be achieved.

- A *goal* is defined [6] as 'something that some stakeholder hopes to achieve in the future'. In L'Ecritoire, it is expressed as a clause with a main verb and several parameters, where each parameter plays a different role with respect to the verb. For example in the goal statement :

  $$'Withdraw_{verb}\ (cash)_{target}\ (from\ ATM)_{means}',$$

  'Withdraw' is the main verb, *'cash'* is the parameter target of the goal, and *'from ATM'* is a parameter describing the means by which the goal is achieved. We adopt the linguistic approach of Fillmore's Case grammar [9], and its extensions [8, 18], to define goal parameters [14]. Each type of parameter corresponds to a case and plays a different role within a goal statement.

- A *scenario* is 'a possible behaviour limited to a set of purposeful interactions taking place among several agents' [6]. In L'Ecritoire, a scenario is defined as composed of one or more *actions* which describe a unique path leading from an *initial* to a *final state* of agents.

  The *initial state* defines the preconditions for the scenario to be triggered. For example, the scenario *'Withdraw cash from the ATM'* cannot be performed if the initial state *'The bank customer has a card'* and *'The ATM is ready'* is not true. The *final state* is the state reached at the end of the scenario. The scenario *'Withdraw cash from the ATM'* leads to the compound state *'The user has cash'*, and *'The ATM is ready'*.

  Actions in a scenario are of two types, atomic actions and flows of actions. *Atomic actions* are interactions '*from'* an agent '*to'* another which affect some '*parameter objects'*. The clause *'The bank customer inserts a card in the ATM'* is an example of an atomic action involving two different agents *'The bank customer'* and *'the ATM'* and having the '*card'* as parameter.

  Flows of actions are composed of several actions and can be of different types, *sequence, concurrent, iterative* and *conditional*. The sentence *'The bank customer gets a card from the bank, then the bank customer withdraws cash from the ATM'* is an example of a sequence comprising two atomic actions. The flow of actions *'While the ATM keeps the card, the ATM displays an "invalid card" message to the bank customer'* is concurrent; there is no predefined order between the two concurrent actions.

- Requirement chunks can be assembled together through *composition, alternative and refinement* relationships. The first two lead to AND and OR structure of RCs whereas the last leads to the organisation of the collection of RCs as a hierarchy of chunks of different granularity.

  AND relationships among RCs link complementary chunks in the sense that every one requires the others to define a completely functioning system. RCs linked through *OR*

relationships represent alternative ways of fulfilling the same goal. RCs linked through a refinement relationship are at different levels of abstraction.

- The L'Ecritoire approach identifies three levels of requirements abstraction, namely the *contextual*, *system interaction* and *system internal* levels.

  The aim of the *contextual level* is to identify the services that a system should provide to fulfil a business goal. At the *system interaction level* the focus is on the interactions between the system and its users to achieve the services assigned to the system at the contextual level. Thus, the contextual level is the bridge between business goals and system functional requirements. The *system internal level* focuses on what the system needs to perform the interactions selected at the system interaction level. The *'what'* is expressed in terms of system internal actions that involve system objects but may require external objects such as other systems. This level defines the software requirements to meet the system functional requirements.

2.2 The scenario-authoring, goal-discovery process

The L'Ecritoire requirements elicitation process is organised around two main activities:

- *goal discovery* and,
- *scenario authoring*

In this process, *goal discovery* and *scenario authoring* are complementary activities, the former following the latter. As shown in Figure 2, these activities are repeated to incrementally populate the requirement chunk hierarchy.

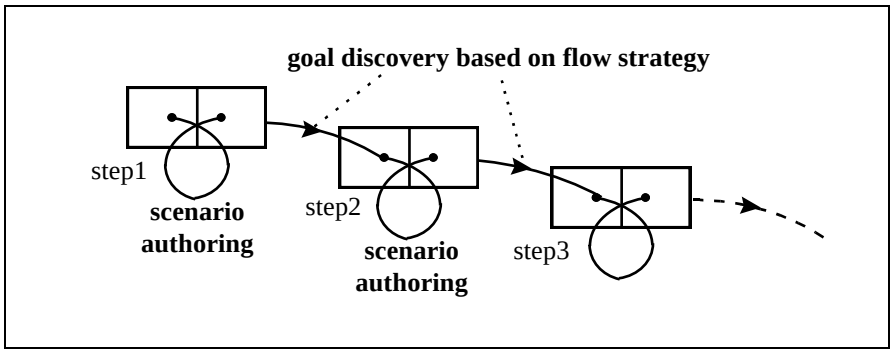

Figure 2: The requirements elicitation process

The requirements elicitation process can be viewed as a flow of steps: each step exploits the goal-scenario relationship in both, the forward and backward directions. A step starts with a goal and the goal-scenario relationship is then exploited in the forward direction to *author* a scenario which is a possible concretisation of this goal. Then the goal-scenario relationship is exploited in the reverse direction to *discover* new goals based on an analysis of the scenario. In subsequent steps, starting from the goals of these new RCs, scenarios are authored and the requirements elicitation cycle thus continues.

Each of the two main activities, goal discovery and scenario authoring, is supported by enactable rules, (1*) authoring rules* and (2) *discovery rules* Authoring rules allow L'Ecritoire.

scenarios which are textual to be authored. Discovery rules are for discovering goals through the analysis of authored scenarios.

2.3 Authoring rules

The role of authoring rules is to ensure the authoring of quality scenarios that can support the automatic goal discovery process. These rules combine *style and contents guidelines* with *linguistic devices*. The former help the L'Ecritoire user in writing scenarios whereas the latter help in scenario analysis, disambiguation and completion. The linguistic devices are based on a case grammar and case patterns. A detailed description can be found in [2,15].

2.3.1 Style and contents guidelines

A L'Ecritoire scenario is a textual one. It is a full prose narrative describing a possible behaviour to fulfil the goal of the associated RC. Style guidelines help the L'Ecritoire user in the wording of the text. " Make use of the active voice", "Avoid the use of pronouns and articles" are examples of style guidelines.

Contents guidelines advise on what is a correct text content. "Any communication action should be directed from an agent to another agent and apply on a parameter" is an example of contents guideline.

Style guidelines are applicable to any type of RCs, contextual, system interaction and system internal whereas there are specific contents guidelines for each level of abstraction. The violation of a style guideline is the sign of an incorrect scenario and can lead to erroneous results when applying the enactable rules. The second style guideline above for example, helps in removing ambiguity whereas the contents guideline helps in writing complete communication statements.

2.3.2 Linguistic devices

Linguistic devices allow the transformation of the initial narrative scenario into a complete, non ambiguous text matching the L'Ecritoire scenario model. This transformation is required to produce a quality requirement document and, also to allow the automatic discovery of goal from scenario analysis using the discovery rules.

Linguistic approach

Scenario transformation is supported by a linguistic approach based on a Case Grammar inspired by Fillmore's Case Theory [9] and its extensions [8, 18]. As shown in Figure 3, the approach is grounded in three elements, *semantic structures*, *semantic patterns* and *scenario model*. Semantic structures correspond to the linguistic structures of statements in the scenario text whereas semantic patterns provide the semantic meaning of these statements. According to Chomsky, linguistic structures are the surface structures of statements whereas semantic patterns correspond to their deep structures. The scenario model provides the structure of concepts of any scenario. The correspondence between linguistic structures and semantic patterns helps in associating a meaning to a scenario statement; that between a semantic pattern and the scenario model defines the relationship between the textual form of a scenario and its conceptual form.

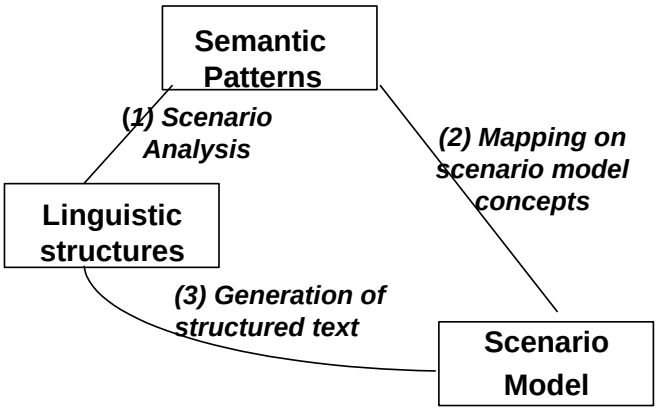

Figure 3 : The linguistic approach for scenario semantic analysis

The semantic analysis of the initial narrative scenario is performed in three steps. The scenario text is parsed and every clause and sentence is matched first onto linguistic structures (1); then onto semantic patterns (2); Thus every clause and sentence of the scenario text is represented as semantic pattern instances. Finally elements of the pattern instances are mapped onto concepts of the scenario model (3).

The example below illustrates the three steps :

Initial clause : ''A card is inserted by a user into the ATM''

Linguistic structure instance: ['A card'](Subject)$_{\text{Object}}$ ['is inserted'](Main Verb)$_{\text{Communication}}$ ['by a user'](Complement)$_{\text{Agent+Source}}$

['into the ATM'](Complement)$_{\text{Destination}}$](VG passive)$_{\text{Communication}}$

Semantic pattern (communication) instance: Communication *('insert') [* Agent *: 'a user' ;*
Object *: 'a card' ;* Source *: 'a user' ;* Destination *: 'the ATM']*

Model concept (Atomic action)
Name *: 'insert';*
To Agent *: 'the ATM' ;*
From Agent *: 'a user' ;*
Parameter *: 'a card';*

Scenario linguistic completion

Linguistic incompleteness of an initial scenario text is detected thanks to the semantic pattern instantiation. When the instantiation of a semantic pattern for a given clause of the initial scenario text is incomplete, L'Ecritoire detects the incompleteness and asks the user to complete the original statement.

For example, for the following scenario action *« a prompt for code is given* » the instantiated pattern *communication(give) [Agent : ? ; object : a prompt for code ; Source : ? ; Destination* : ?] shows missing elements. Every '?' above must be replaced by a term. This leads to the completed sentence : *« a prompt for code is given by the ATM to the user »*

Scenario linguistic disambiguation

The syntactical analysis previous to pattern matching is used to detect anaphoric references in the initial scenario text and to ask the user to replace them by unambiguous terms.
For example in the action *« the user inserts his card in the ATM »*:

the anaphoric reference ‘his’ is detected; the user is asked to replace ‘his’ by a non-ambiguous term, the ‘user’, and the action is rephrased as *« the user inserts the user’s card in the ATM »*.

2.4 Discovery rules

Discovery rules guide the L’Ecritoire user in discovering new goals and therefore, eliciting new requirement chunks. The discovery is based on the analysis of scenarios through one of the three proposed discovery strategies, namely the *refinement*, *composition* and *alternative* strategies. These strategies correspond to the three types of relationships among RCs introduced in section 2.1 above. Given a pair <G,Sc>:

- the composition strategy looks for goals Gi ANDed to G,
- the alternative strategy searches for goals Gj ORed to G,
- the refinement strategy aims at the discovery of goals Gk at a lower level of abstraction than G.

Therefore, *composition* (*alternative*) *rules* help in discovering ANDed (ORed) goals to G. These are found at the same level of abstraction as G. The <G,Sc> chunk is processed by the *refinement rules* to produce goals at a lower level of abstraction than G. This is done by considering (in a similar way to that suggested by Cockburn [4]) each interaction in Sc as a goal. Thus as many goals are produced as there are interactions in Sc.

As shown in Figure 4, once a complete scenario has been authored, any of these three strategies can be followed. Thus, there is no imposed ordering on the flow of steps which instead, is dynamically defined.

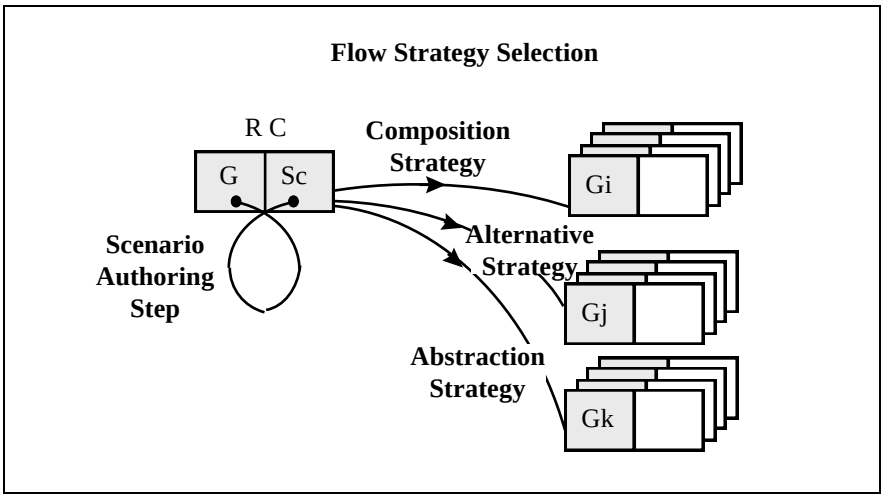

Figure 4: Selecting a strategy

L’Ecritoire uses six discovery rules, two for each strategy. Rules can be applied at any of the three levels of abstraction, contextual, system interaction and system internal. A detail description of rules can be found in [16, 19]. As an example of a rule, we present the refinement rule R1 and exemplify it with the example of ATM system engineering.

Refinement guiding rule (R1)

Goal :	Discover (from requirement chunk <G,Sc>)$_{So}$ (goals refined from G)$_{Res}$ (using every atomic action of Sc as a goal)$_{Man}$
Body :	Step1 : Associate a goal Gi to every atomic action Ai in Sc. Gi refines G
	Step2 : Complement Gi by the manner 'in a normal way'
	Step3 : User evaluates the proposed panel of goals Gi and selects the goals of interest
	Step4 : Requirement chunks corresponding to these selected goals are ANDed to one another

The guiding rule R1 aims at refining a given requirement chunk *(from RC<G,Sc>)$_{So}$* by suggesting new goals at a lower level of abstraction than G *(goals refined from G)$_{Res}$*.

The refinement mechanism underlying the rule looks to every interaction between two agents in the scenario Sc as a goal for the lower level of abstraction (step1). Let us take as an example the scenario of the requirement chunk RC presented below:

Goal G: Improve services to our customers by providing cash from the ATM

Scenario SC :

1- If the bank customer gets a card from the bank,

2- Then, the bank customer withdraws cash from the ATM

3- and the ATM reports cash transactions to the bank.

This scenario text corresponds to the structured textual form of the scenario as it results from the authoring step. The internal form is a set of semantic pattern instances which clearly identify three agents namely, the bank, the customer and the ATM as well as three interactions namely 'Get card', 'Withdraw cash' and 'Report cash transactions' corresponding to the three services involving the ATM. These services are proposed as goals of a finer grain than G, to be further made concrete by authoring scenarios for these goals.

We propose that these scenarios describe the normal course of actions. Thus, the manner parameter of every generated goal Gi is fixed to *'in a normal way'* (step2). This leads in the above example, to propose to the user the three following refined goals :

- *'Get card from the bank in a normal way'*
- *'Withdraw cash from ATM in a normal way'*
- *'Report cash transactions to the bank in a normal way'*

Assuming that the user accepts the three suggested goals (step3), the corresponding requirement chunks are ANDed to one another (step4).

3. The Usage Scenario

In this section, we illustrate the functionality of the L'Ecritoire tool through a usage scenario drawn from the ATM system. The scenario shows how the user of the tool interacts with L'Ecritoire to engineer system requirements. It illustrates a top down approach which starting from a goal at the system interaction level and using the three discovery strategies of section 2, guides the elicitation of software system requirements that will eventually be matched to Lyee software requirements.

Step 1: Starting the Session

The session starts with the definition of the domain glossary. This can be done either by introducing a new glossary or by reusing one from L'Ecritoire's repository of domain glossaries. As shown in Figure 5 the glossary classifies terms into three linguistic categories, verbs, agents, and objects. A 'Check Vocabulary ' button is used to help detect synonyms in the glossary. For example, the terms 'cash' and 'money' will be detected as synonyms in the object list and one can be removed. Synonym detection helps in eliminating redundancy in the glossary.

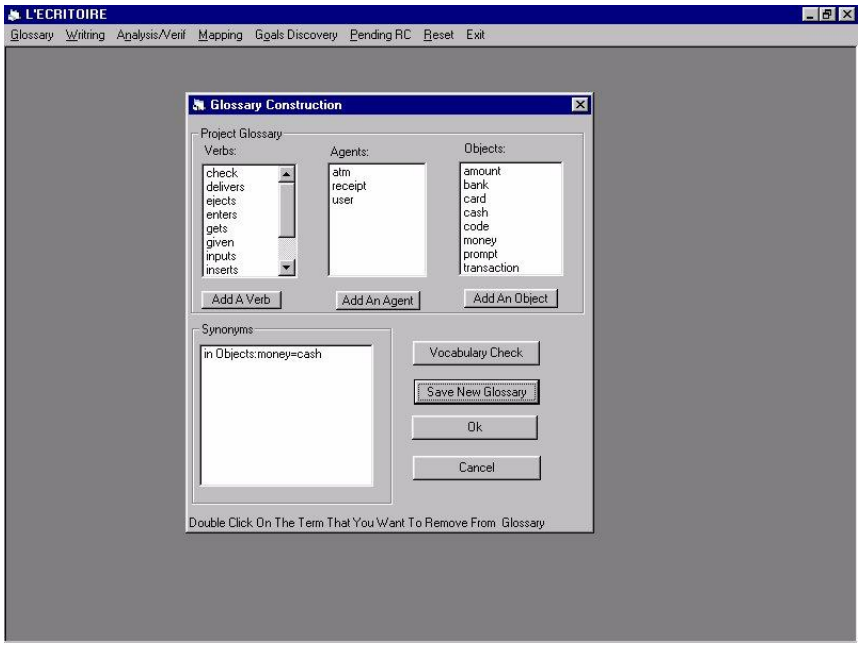

Figure 5: The glossary window

Step 2: Capturing a new RC

There are two activities in capturing a new RC (a) entering the new goal and (b) authoring a scenario for this goal. Both these can be done in the window presented in Figure 6.

(a) The user enters the goal *'Withdraw cash from the ATM in a normal way'*. The user has also to name the associated RC (Withdraw 1) and provide its type (interaction).

(b) A scenario is authored for the goal entered in (a) above. Since the scenario is entered in natural language, the user may use the facilities provided by the four buttons, Consult Glossary, Contents Guidelines, Style Guidelines, and Check Spelling. The first allows the user to consult the glossary to use the appropriate term. The second and the third provide content and style guidelines appropriate for this type of RC. Finally, it is possible to perform a spell-check on the scenario being authored using the last button.

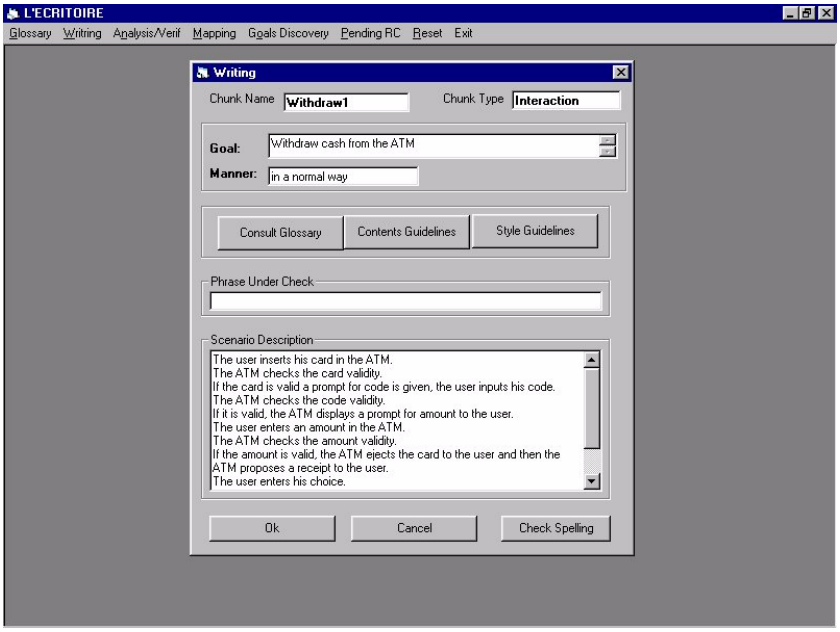

Figure 6: Capturing a new RC

The authored scenario is reproduced below for ease of reference in the rest of this paper.

The user inserts his card in the ATM.
The ATM checks the card validity.
If the card is valid , a prompt for code is given, the user inputs his code.
The ATM checks the code validity.
If it is valid , the ATM displays a prompt for amount to the user.
The user enters an amount.
The ATM checks the amount validity.
If the amount is valid , the card is ejected and then the ATM proposes a receipt to the user.
The user enters his choice.
If a receipt was asked, the receipt is printed but before the ATM delivers the cash.

Step 3: Scenario Analysis

L'Ecritoire provides a guidance mechanism to progress in the process. This mechanism is activated by clicking the right button of the mouse. It gives a menu of rules that can be enacted upon the requirement chunk under consideration. Using this progress guidance mechanism, the user may decide to progress with the guided analysis of the scenario just captured. The Analysis and Verification window is shown in Figure 7. It appears with the textual scenario written during the previous step displayed in the text zone of the top left of the window.

The actions that can be performed on this scenario with this window can be classified into two groups, glossary actions and linguistic actions. The former allow glossary update whereas the latter triggers the linguistic devices.

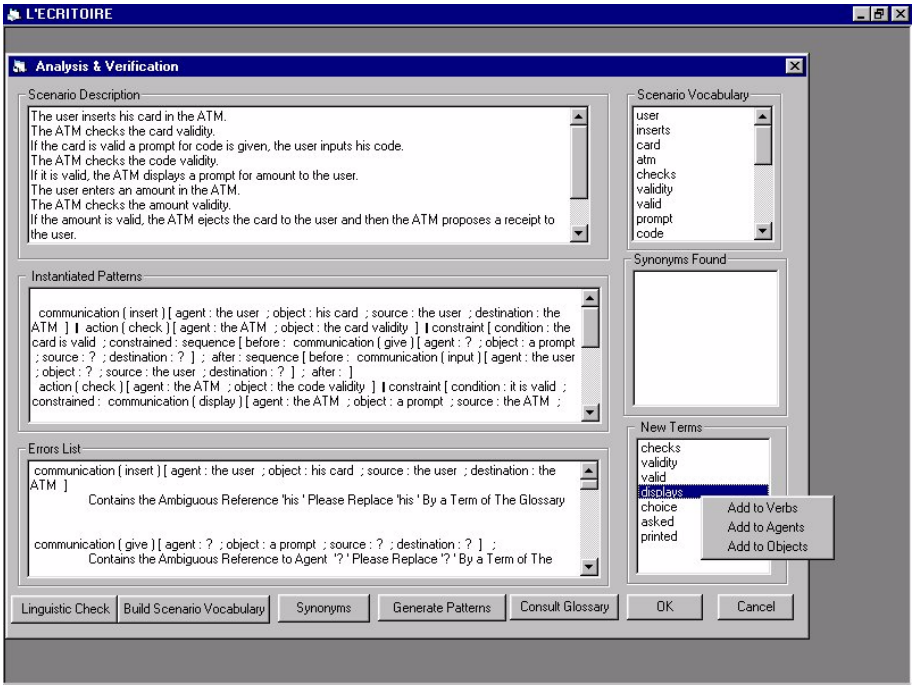

Figure 7: The analysis and verification window

The three buttons, Synonyms, Consult Glossary, and Build Scenario Vocabulary trigger glossary actions. When the 'Consult Glossary' button is clicked then the scenario text is parsed to identify the terms already available in the glossary and those which are new. The former terms are displayed in the Scenario Vocabulary widget whereas the latter are displayed in the New Terms widget. The user may decide to enter the new terms in the Glossary using the Build Scenario Vocabulary button. As shown in Figure 7, the entry shall be made in the appropriate linguistic category. Synonym detection can be performed using the Synonym button. Terms used in the scenario which are synonyms of Glossary Terms are detected and put up in the Synonym Found widget for possible replacement.

To activate the linguistic devices, the user has to click the Generate Patterns button of Figure 7. The result of the linguistic analysis is shown as a set of instantiated patterns in the Instantiated Patterns widget.

A list of errors found during pattern instantiation can be displayed in the Error List widget by clicking the button Linguistic Check. The errors reported in the Error List must be removed from the scenario. These corrections are performed in the Scenario Description widget and step 3 as described here is repeated till no errors are found. The corrected scenario is finally found in the Scenario Description widget. This scenario is stored in the L'Ecritoire repository.

The corrected version of the initial scenario of step 2 is shown below with the corrections in bold.

The user inserts **a card in the ATM.**
The ATM checks the card validity.
If the card is valid a prompt for code is given **by the ATM to the user**, the user inputs the code **in the ATM**.
The ATM checks the code validity.
If **the code is valid**, the ATM displays **a prompt for amount to the user**.
The user enters an amount **in the ATM**.

The ATM checks the amount validity.
If the amount is valid, **the ATM ejects the card to the user** and then the ATM proposes a receipt to the user.
The user enters **the user's choice** in the ATM.
If a receipt was asked the receipt is printed **by the ATM to the user** but before the ATM **delivers the cash to the user.**

Step 4: Structuring Scenario Text

The instantiated patterns corresponding to the text above represent the essence of the scenario as input by the user. Whereas these patterns can be used by L'Ecritoire internally, they are not in a readable form for the user. The purpose of step 4 is to map the patterns instances into a structured readable text.

Clicking the OK button of the previous window prompts the window of Figure 8 with its Instantiated Patterns widget containing the instantiated patterns. When the Map button is pressed then L'Ecritoire generates the structured text for the scenario and displays it in the Structured Text widget.

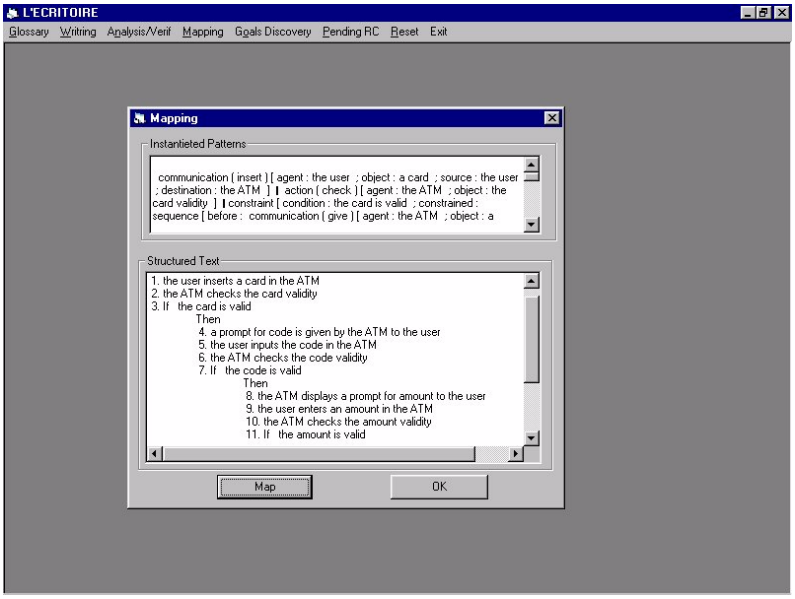

Figure 8: The Mapping window

The structured scenario is reproduced below for sake of clarity.

1. the user inserts a card in the ATM
2. the ATM checks the card validity
3. If the card is valid
   Then
   4. a prompt for code is given by the ATM to the user
   5. the user inputs the code in the ATM
   6. the ATM checks the code validity
   7. If the code is valid
      Then
      8. the ATM displays a prompt for amount to the user
      9. the user enters an amount in the ATM
      10. the ATM checks the amount validity

11. If the amount is valid
    Then
    12. the ATM ejects the card to the user
    13. the ATM proposes a receipt to the user
    14. the user input the user's choice
    15. If a receipt was asked
        Then
        16. the ATM delivers the cash to the user
        17. the receipt is printed by the ATM to the user

Step 5: Discover Goals From the Scenario

Upon using the progress guidance mechanism again, the user is presented with the three strategies, Refinement, Complementary, and Alternative, that can be deployed for goal discovery from the scenario. The user considers the properties of the scenario to decide the strategies to be adopted. Evidently, the scenario above has a large number of variants as shown by the number of If statements in it. Therefore the Alternative strategy must be deployed. Besides, an examination of the first three conditions of the scenario (lines 3, 7 and 11) shows that there are many ways in which these could be handled. Therefore a systematic elicitation of internal requirements should be performed. This can be achieved by the deployment of the Refinement strategy.

Having determined the two interesting strategies, the user decides to first select the Alternative strategy and then the Refinement one. The response of L'Ecritoire is shown in steps 6 and 7 respectively

Step 6: Using the Alternative Strategy

In our illustration, the user selects the alternative strategy and within this strategy the rule to *Search alternative manners to fulfil the same goal*. The window of Figure 9 now appears on the screen with the Conditions List and Missing Cases widgets already filled in. The former contains the conditions of the If statements of the scenario of step 4. The ordering of the conditions reflects the nesting of the If statements in the scenario. This rule computes all possible combinations of the negation of these conditions and considers each of these as a missing case. These cases are displayed in the Missing Cases Widget. As shown in Figure 9, there are four conditions C1 to C4 in the scenario. The rule computes the four missing cases, not C1, C1 and not C2, C1 and C2 and not C3, C1 and C2 and C3 and not C4.

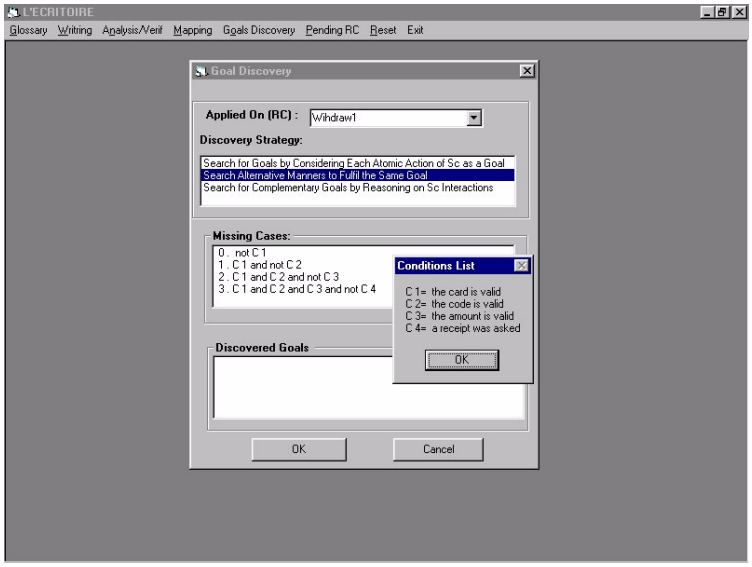

Figure 9: The goal discovery window for the alternative strategy

The user now examines each of the candidate missing cases and if found relevant, he formulates it as a goal. As soon as the user selects a case in the Missing Cases widget, the Discover window is prompted as shown in Figure 10. Thus the user can formulate the goal corresponding to this missing case. As illustrated in Figure 10, the case, C1 and not C2, is found relevant. For the ATM this means that it must react to a wrongly entered PIN code and so the user formulates the goal, *Withdraw cash with an error correction phase* (Figure 10).

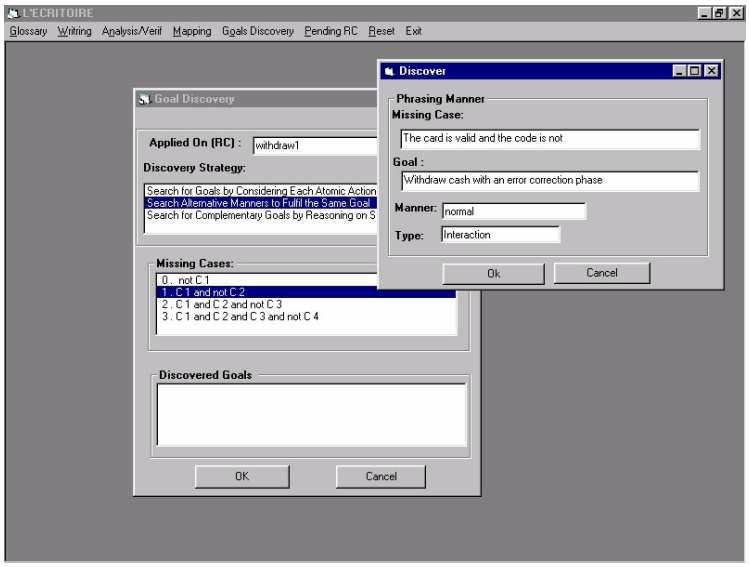

Figure 10: The goal discovery window for the alternative strategy

Step 7: Using the Refinement Strategy

Let us now consider the second case, that of selection of the Refinement strategy, of step 5 above. The window shown in Figure 11 appears with the Action List widget already displayed. The list of actions corresponds to the atomic actions of the scenario which have been automatically extracted from the internal representation of the scenario stored in the L'Ecritoire repository. The user considers each action in this list as a candidate goal at the next lower abstraction level. Upon selection of such an action, the

Discover window is prompted to allow rephrasing it as a goal. The type of the RC associated to this goal is automatically set to "internal". As shown in Figure 11, action 2 of the scenario of step 4

2. the ATM checks the card validity

is refined as the goal *Check the card validity*. Similarly for action 6 the goal is *Check the code validity*.

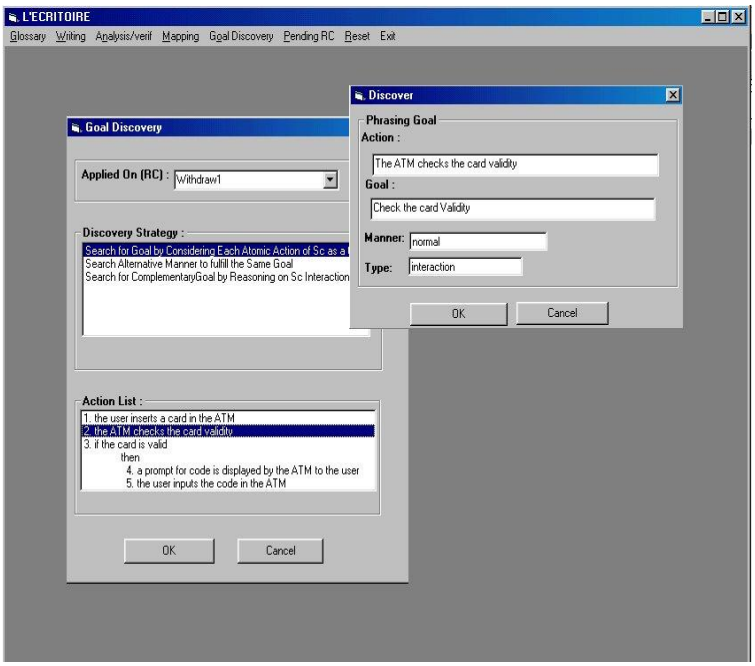

Figure 11: The goal discovery window for the refinement strategy

Terminating the Session

At this point the user has discovered three goals, namely, *Withdraw cash with an error correction phase*, *Check the card validity*, and *Check the code validity*. A scenario has still to be authored for each one of these from which new goals will be discovered. It can be done in this session itself or in another session. In the latter case, the current session is terminated. When a new session will be launched then L'Ecritoire will display the list of pending RCs as shown in Figure 12.

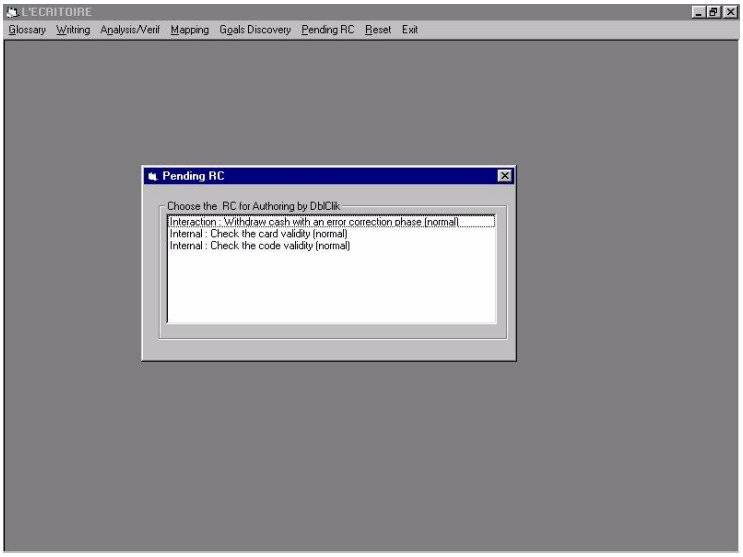

Figure 12: The pending RCs

4. Discussion of the L'Ecritoire-LyeeALL Coupling

In this section we discuss the L'Ecritoire-LyeeALL coupling with the view to showing the benefits that accrue from it. LyeeALL is a program generation tool which produces high quality, reliable, and stable code from a set of software requirements formulated in LyeeALL terms. Seeing its downstream nature, need for upstream support was felt and L'Ecritoire provides it. The contribution of L'Ecritoire to the L'E-Lyee project is fourfold:

- Automatic production of software requirements in Lyee terms,
- Fitting Lyee software requirements to business goals,
- Achieving completeness and consistency of Lyee software requirements, and
- Requirement traceability

We consider each of these in turn.

(1) Producing software requirements

Broadly speaking, the formulation of Lyee software requirements can be done in terms of four elements, unit, word, formula, and ordering. A *unit* is a set of *words* that is manipulated during an interaction between the user and the software system. Words can be input, when their values are given by the user, or output, when they are computed by the system. For each output word, there must be a *formula* to calculate its value. An *ordering* between units captures a precedence relationship between unit execution.

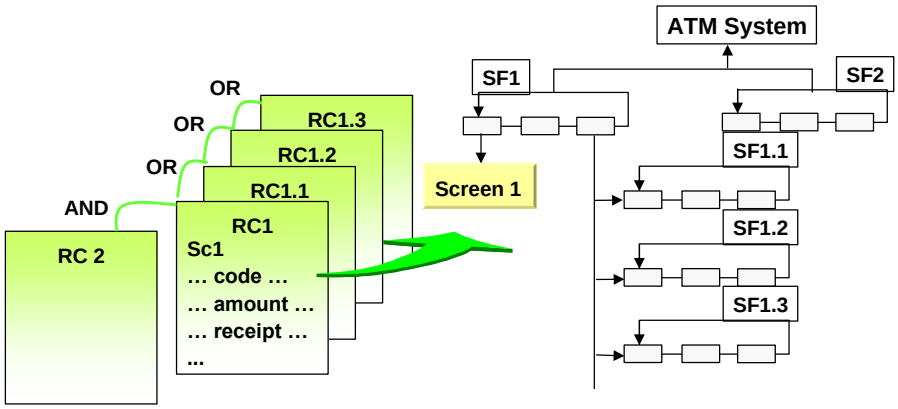

Figure 13: A simplified L'Ecritoire-Lyee correspondence

The meeting point between L'Ecritoire requirements and Lyee requirements is at the system interaction level (section 2). A simplified view of this meeting point is shown in Figure 13. As highlighted, a correspondence can be established between a system interaction RC and a Lyee unit. Furthermore, the scenario of such an RC provides the words of the Lyee unit. Finally, the AND/OR relationship between system interaction RCs forms the basis for the ordering between Lyee units. As shown in Figure 13, each Lyee unit is associated to a so-called Scenario Function, SF, which is the program execution control mechanism to capture the input and produce the output of the unit. Ordering between units is graphically represented by AND/OR links between SFs. For instance, in Figure 13, there is an AND link between SF1 and SF1.1 whereas there is an OR link between SF1.1, SF1.2, and SF1.3. Broadly speaking a correspondence can be established between AND/OR RC relationships and AND/OR SF links.

By automating the foregoing correspondence, the coupling of L'Ecritoire to LyeeALL allows code generation from user-centric requirements expressed in natural language through goal-scenario tuples.

(2) Fitting Lyee requirements to business goals

The second key benefit is the close fit between Lyee software requirements and the business goals of the organisation in which the software will operate. This fit is on account of the refinement process which moves down the contextual, system interaction, and system internal levels of abstraction. This ensures that every system interaction requirement flows from recognised and agreed business goals. Further, every system internal requirement is derived from system interaction requirements and therefore meets the business goals.

The use of the alternative strategy within this refinement process suggests a systematic exploration of different alternatives at each of the three levels. This can be seen as a search for the best fit between software system requirements and business goals.

(3) Completeness and Consistency

L'Ecritoire ensures that the collection of software system requirements that it produces is complete and consistent. The three strategies help in achieving this completeness. The alternative and composition rules ensure completeness at a given level of abstraction. The former do this by identifying all the variants of a scenario whereas the latter help in identifying all the complementary scenarios of a given one. Once completeness has been

ensured at a given level, the refinement rules propagate it down the different levels of abstraction of L'Ecritoire, thus, ensuring completeness at the system interaction and internal levels.

The consistency of the collection of requirements is largely due to the linguistic treatment of scenarios. This treatment first ensures the semantic consistency of scenarios and then guarantees that the goals generated from the scenarios are valid ones.

This property of L'Ecritoire gives to LyeeALL a complete and consistent collection of requirements. The result is a reduction of the number of iterations of LyeeALL to generate code. In the absence of L'Ecritoire, these iterations were required to align the generated program to organisational needs.

(4) Traceability

The L'Ecritoire-LyeeALL coupling achieves complete traceability, pre-traceability and post-traceability. L'Ecritoire provides the former whereas LyeeALL provides the latter. Pre-traceability is obtained by the hierarchy of requirement chunks which links business goals to system internal requirements through system interaction requirements. The documentation associated with the hierarchy of chunks facilitates the propagation of changes in business goals down to the software system requirements. The Lyee program generation mechanism integrates in itself a post-traceability capability. A change in the software requirements is automatically propagated to programs through code regeneration.

5. References

[1]A.I. Anton, *Goal based requirements analysis*. Proceedings of the 2nd International Conference on Requirements Engineering ICRE'96, pp. 136-144, 1996.

[2] C. Ben Achour, *Requirements Extraction From Textual Scenarios.* PhD Thesis, University Paris6 Jussieu, January 1999.

[3] J. Bubenko, C. Rolland, P. Loucopoulos, V De Antonellis, *Facilitating 'fuzzy to formal' requirements modelling*. IEEE 1st Conference on Requirements Enginering, ICRE'94 pp. 154-158, 1994.

[4] J.M. Caroll, *The Scenario Perspective on System Development*, in J.M. Carroll (ed.), Scenario-Based Design: Envisioning Work and Technology in System Development (1995).

[5] Cockburn, *Structuring use cases with goals*. Technical report. Human and Technology, 7691 Dell Rd, Salt Lake City, UT 84121, HaT.TR.95.1, http://members.aol.com/acocburn/papers/usecases.htm (1995).

[6] CREWS Team, *The CREWS glossary*, CREWS report 98-1, http ://SUNSITE.informatik.rwth-aachen.de/CREWS/reports.htm.

[7] A.M. Davis, *Software requirements :objects, functions and states*. Prentice Hall, 1993.

[8] S.C. Dik, *The theory of functional grammar, part I : the structure of the clause*. Functional Grammar Series, Fories Publications, 1989.

[9] C. Fillmore, *The case for case*. In ''Universals in linguistic theory'', Holt, Rinehart and Winston (eds.*)*,Bach & Harms Publishing Company, pp. 1-90, 1968.

[10] A.van Lamsweerde, *Goal-Oriented Requirements Engineering, : A guided tour,* Mini tutorial on RE, 5th IEEE International Symposium on Requirements, Engineering, pp249-264, Toronto, Canada, 2002

[11] R.L. Mack, *Discussion : Scenarios as Engines of Design*, in John M. Carroll (ed.), Scenario-Based Design: Envisioning Work and Technology in System Development (John Wiley and Sons, 1995) 361-387.

[12] : C. Potts, K. Takahashi, A.I. Anton, *Inquiry-based requirements analysis*. In IEEE Software 11(2), pp. 21-32, 1994.

[13] : C. Potts, *Fitness for use : the system quality that matters most.* Proceedings of the Third International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'97 , Barcelona, pp. 15-28, June 1997.

[14] N. Prat, *Goal formalisation and classification for requirements engineering*. Proceedings of the Third International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'97, Barcelona, pp. 145-156, June 1997.

[15] C. Rolland, C. Ben Achour, *Guiding the construction of textual use case specifications*. Data & Knowledge Engineering Journal Vol. 25 N° 1, pp. 125-160, (ed. P. Chen, R.P. van de Riet) North Holland, Elsevier Science Publishers. March 1997.

[16] C. Rolland, C. Souveyet, C. Ben Achour, *Guiding Goal Modelling using Scenarios*. IEEE Transactions on Software Engineering, Special Issue on Scenario Management, Vol. 24, No. 12, Dec. 1998**.**

[17] C. Rolland, G. Grosz, R. Kla, *Experience With Goal-Scenario Coupling.* In Requirements Engineering, Proceedings of theFourth IEEE International Symposuim on Requirements Engineering, Limerik, Ireland, 1999

[18] R.C. Schanck, *Identification of conceptualisations underlying natural language*. In ''Computer models of thought and language'', R.C. Shanck, K.M. Colby (eds), Freeman, San Francisco, pp. 187-247, 1973.

[19] M. Tawbi, *Crews-L'Ecritoire : un guidage outillé du processus d'Ingénierie des Besoins*. Ph.D. Thesis University of Paris 1, October 2001.