CREWS Report Series 98-1

GUIDING THE CONSTRUCTION OF TEXTUAL USE CASE SPECIFICATIONS

Colette Rolland, Camille Ben Achour

CRI, Université de Paris I Panthéon-Sorbonne, 17 rue de Tolbiac, 75013 Paris, France

{rolland, camille}@univ-paris1.fr

Appeared in Data & Knowledge Engineering Journal, Vol 25, N°1-2, pp. 125-160, (ed. P. Chen, R.P. van de Riet), North Holland, Elsevier Science Publishers. March 1998.

Guiding the construction of textual use case specifications[1]

Colette Rolland, Camille Ben Achour

CRI, Université de Paris I Panthéon-Sorbonne, 17 rue de Tolbiac, 75013 Paris, France

Abstract

An approach for guiding the construction of use case specifications is presented. A use case specification comprises contextual information of the use case, its change history, the complete graph of possible pathways, attached requirements and open issues. The proposed approach delivers a use case specification as an unambiguous natural language text. This is done by a stepwise and guided process which progressively transforms initial and partial natural language descriptions of scenarios into well structured, integrated use case specifications. The basis of the approach is a set of linguistic patterns and linguistic structures. The former constitutes the deep structure of the use case specification whereas the latter corresponds to the surface structures. The paper presents the use case model, the linguistic basis and the guided process along with the associated guidelines and support rules. The process is illustrated with the automated teller machine (ATM) case study.

Keywords : Textual Scenario Analysis, Use Case Specification, Requirements Elicitation

1. Introduction

Scenarios have been gaining increasing popularity in both Human Computer Interaction (HCI) and Software Engineering (SE) as ‘engines of design’ [3, 25]. In HCI scenarios are used to focus discussion on usability issues [14, 27, 45].They support discussion to gain an understanding of the goals of the design [23] and help to set overall design objectives [20, 34]. In contrast, scenarios play a more direct role in SE, particularly as a front end to object oriented design [8, 22, 29, 31, 35, 36, 37, 38, 47]. Use case driven approaches [6, 8, 15, 16, 17, 30] have proved useful for requirements elicitation and validation. The aim of use cases in Requirements Engineering is to capture systems requirements. This is done through the exploration and selection of system-user interactions to provide the needed facilities. A use case is a description of one or more end to end transactions involving the required system and its environment [28]. The basic idea is to specify use cases that cover all possible pathways through the system functions [8, 30, 35]. The concept of use case was originally proposed in Objectory [15, 16] but has recently been integrated in a number of other approaches including the Fusion method [7] and the Unified Modeling Language [38].

[1] The work presented in this paper is funded by the European Community within the framework of the ESPRIT LTR project CREWS (*Cooperative Requirements Engineering With Scenarios*) n°21903.

The literature shows that scenarios come in different forms : textual, animated, prototypes etc. However, a number of authors [12, 17, 19, 21, 26, 27, 39, 42] recommends textual ones. A survey of scenario based approaches conducted within the CREWS project [32] has shown that over thirteen approaches eight recommend the use of ‘natural language text descriptions of system use’. The widespread use of textual scenarios was also confirmed by visits of industrial sites which showed that, in practice, 90% of scenarios are expressed in either textual form or formatted text. The main reason seems to be that natural language provides a way to express the problem in an easy to understand representation [20]. It is a means to manage the tension between informal expressions of users and more formal design representations and requirements specifications typical of requirements engineering and software engineering.

All the foregoing approaches start off by assuming that use cases are given. Thus, there is no interest in discovering how these are constructed, what is their structure, how can one ensure their consistency and completeness etc. However these are important questions because if use cases are not properly constructed, are ambiguous, incomplete, or inconsistent then the interactions they specify will also inherit these properties. The lack of guidelines to support the process of constructing use cases is certainly one of the drawbacks of use case driven approaches for requirements engineering [44].

In this paper we propose an *approach to guide the construction of textual use case specifications*. Our approach treats a use case specification as a complex structure comprising information about the usage context of the use case, the complete graph of possible pathways for normal and exceptional courses of actions, attached requirements and open issues. In order to deal with the complexity of the use case structure the proposed approach organizes its construction through an incremental process as depicted in figure 1. The process is supported by rules which guide the transformation of short narrative stories called scenarios and their consistent integration into complete episodes of the use case specification.

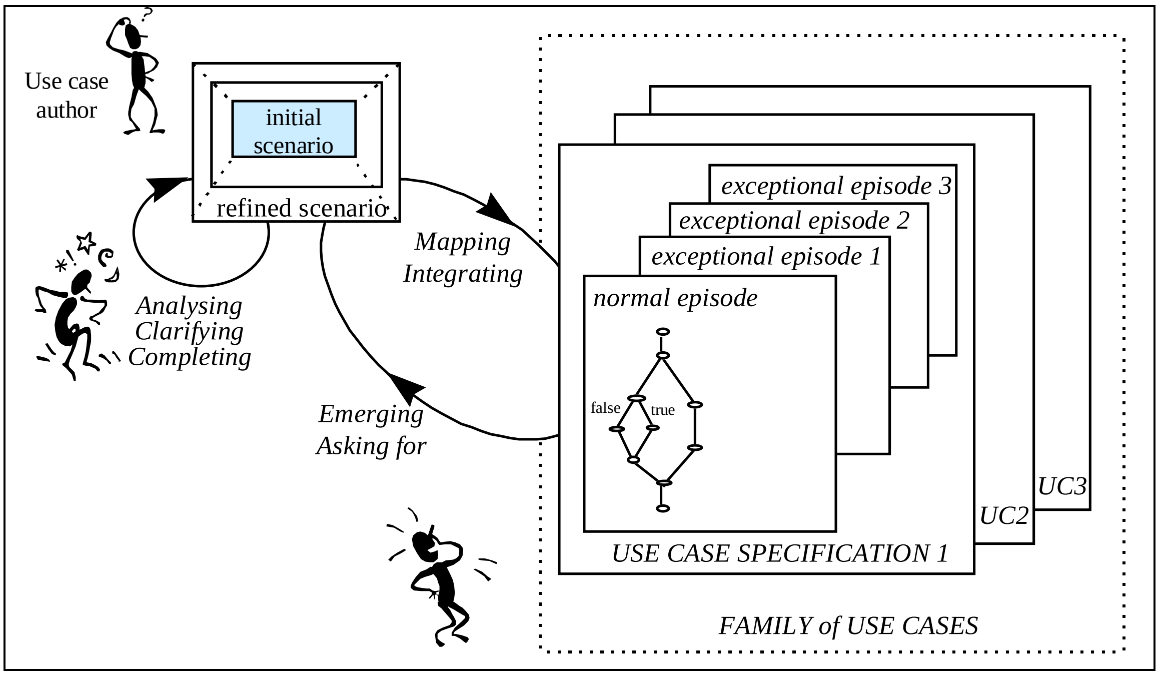


Figure 1 : Incremental guided process of use case specification.

Construction, in our approach, means looking at completeness and consistency issues on one hand, and readability and communication issues on the other hand. In order to support the former the process is driven *by enactable rules* whereas support for the latter is through *natural language*. Because of the nature of the construction process, the guidance provided is twofold :

- *linguistic guidance* to support the expression of the use case specification using natural language and,

- *construction guidance* to support the consistent integration of partial descriptions of interactions into a complete use case specification.

By combining linguistic and construction guidance, the proposed approach aims at :

(1) delivering a non ambiguous text written in structured natural language which constitutes the use case specification and,

(2) supporting its production by a stepwise and guided process which progressively transforms initial and partial scenario descriptions in natural language prose into well structured and non ambiguous texts integrated in the use case specification.

The advantage of the former is an easier communication among the domain experts and the requirements engineers since natural language is used. The latter enables integration of scenarios into complete use case specifications. This is more easily done through our approach than through manual means.

The choice of the textual form of use case specification calls for natural language support and makes natural language analysis critical. However, the use of natural language in a completely free mode increases the risks of ambiguity, inconsistency and incompleteness. These risks [1, 4, 33] make the automatic interpretation of natural language texts complex and difficult to achieve. The use of clause templates [6] is another possibility but we believe that it undoubtedly constrains the behavior of the use case author. Our approach looks for the middle ground between these two and combines the use of narrative prose to express scenarios with structured natural language for the use case specification. Besides, in order to handle the linguistic aspect of the work, the use case author is guided in two ways. First, the writing of scenarios is guided by *style and contents guidelines* which help :

- limiting the size of input scenarios,
- targeting their contents,
- providing, if necessary, templates to write sentences,
- advising the scenario author of what is expected at this very moment in the process.

Second, there are *rules* for the analysis, clarification and completion of natural language scenario sentences which support the mapping of the initial sentences onto the structured clauses of the use case specification. This permits us to take advantage of free text expression while limiting the complexity of their interpretation.

The linguistic basis for these guidelines and rules is a set of *linguistic patterns* and *linguistic structures*. These are connected to the precise semantics that we defined for a use case. The linguistic patterns are generic and represent the deep structure [5] of the use case specification. The linguistic structures correspond to the surface structures [5] and give the multiple external forms of sentences corresponding to the same generic pattern.

The remaining of the paper is organized as follows. The use case model is presented in section 2. We deal with its semantics in section 3 by detailing the linguistic patterns and linguistic structures. Section 4 gives an overview of the guided process and section 5 illustrates the process with the ATM (Automated Teller Machine) example. Finally we draw some conclusions and identify future work in section 6.

2. Use case model

The proposed use case model is a compilation of information from three sources. The first one is the literature and particularly the works [6, 8, 30, 24, 38]. The second one is the industrial visits made to CREWS industrial committee members in 1997 [18] and the third source is the joint work developed in CREWS on knowledge representation for scenarios based approaches [11]. We shall describe the use case model in three steps : the contextual information of a use case first, then its contents and finally the natural language expression of its contents. These are highlighted in figure 2, 3 and 4, respectively. The model is presented using binary E/R like notations.

2.1 Use case contextual information

The role of the contextual information attached to a use case specification is to situate the agent/system interactions described in the use case within the organisational setting and goals. Figure 2 presents the contextual information we associate to a use case specification. We shall introduce the various elements progressively.

A use case involves an *initiating agent* who has in mind a *goal* of service delivered by a system. The purpose of the use case specification is to describe how the agent can interact with the system to get the service and achieve his/her goal. For instance, in the case of cash withdrawal with an ATM, the initiating agent is the user of the ATM who wants to get cash.

In order to fulfil his/her goal the initiating agent interacts with other agents that we call *secondary agents*. An agent may be a human or a machine ; the system itself (for instance the ATM) is an agent. In the ATM use case, the bank of the user may be a secondary agent that will be part of the use case specification. Similarly, the process of fulfilling a user goal which is the core of the use case specification may refer to *resources*. A resource can be a physical resource such as a credit card or an abstract resource such as a user account.

There are some conditions required for the interactions to start. Such conditions describe the *initial state* and are expressed in terms of agent states. For instance, the interactions of the ‘withdraw cash’ use case does not start if the following condition does not hold : ‘’*the user has a card and the ATM is ready*’’.

In order to make the external view of the system as complete as possible, there is a need for a *family* of use cases. For example, the *ATM application* family comprises the three use cases *withdraw cash*, *fill in the ATM* *with cash* and *fill in the ATM with receipt paper*. A *glossary* of terms is maintained in relationship with the family of use cases.

The use case specification may correspond to a *design alternative* which is one among several other design solutions envisioned to fulfil a high level goal of the organisation. For example, if providing better services to bank customers is the goal of a pool of banks, '*withdraw cash with the ATM*' might be one design alternative against other ones such as '*front desk servicing*' and '*audio teller machine*'.

Finally, the use case may be connected to *open issues* and *system requirements.* Both arise when the process of specifying the use case proceeds. An open issue is a pending question such as '*How does the user get his card back if the ATM keeps it?*'. System requirements other than the ones which are formalised in the use case specification itself may emerge in the course of the specification process. '*The ATM must have a clock*' or '*the ATM must maintain the list of cash withdrawals performed during the last seven days*' are examples of such requirements.

Finally, the use case contextual information is the following :

- use case *name*,
- use case *family*,
- *initiating agent* and his/her associated *goal*,
- *initial state* : the precondition for the use case course of actions to be triggered,
- *design alternative,*
- *secondary agents* : those who participate in the use case other than the initiating agent,
- *resources* used in the use case.

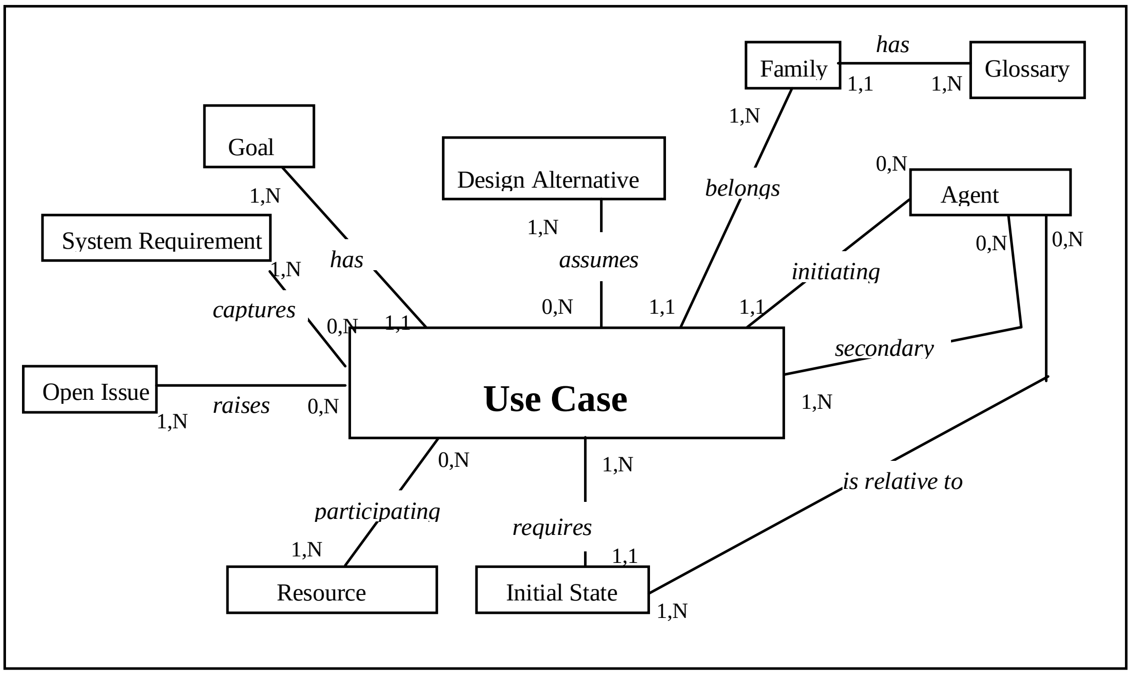
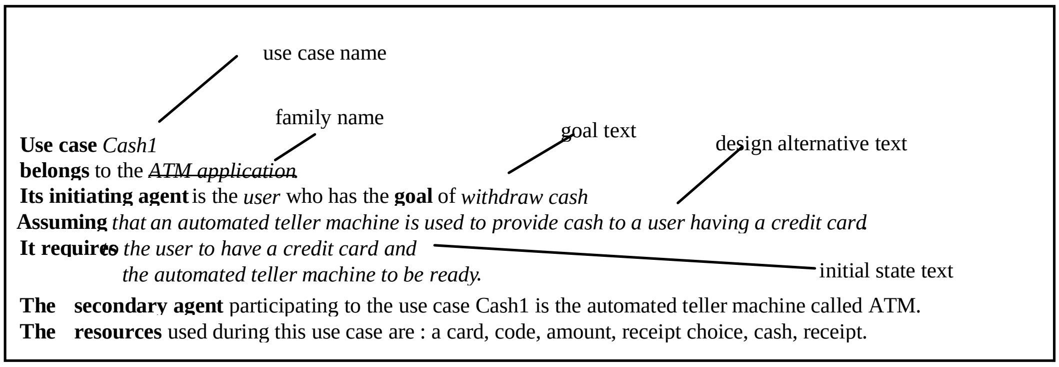


Figure 2 : Use case contextual information.

The textual form of the contextual information for the 'withdraw cash' example is presented below.


Use case *Cash1*
belongs to the ~~*ATM application*~~.
Its initiating agent is the *user* who has the **goal** of *withdraw cash*
Assuming *that an automated teller machine is used to provide cash to a user having a credit card*
It requires *to the user to have a credit card and*
the automated teller machine to be ready.
The secondary agent participating to the use case Cash1 is the automated teller machine called ATM.
The resources used during this use case are : a card, code, amount, receipt choice, cash, receipt.

2.2 Use case contents

The core concept of the use case model is the concept of *action*. An action may be an *atomic action* or a *flow of actions*. An atomic action materialises an interaction *from* an agent *to* another agent and requires some resources. By contrast a flow of actions is a complex action composed of other actions connected through predefined constructors : *sequence*, *concurrency*, *iteration* and *alternative*. An *episode* is composed of a flow of actions and of several possible *final states* reachable from the flow of actions. The contents of a use case is composed of a *normal episode*, and zero, one or several *exceptional episodes.* The normal episode specifies the normal courses of actions permitting the agent to fulfil his/her goal whereas exceptional episodes fail to achieve his/her goal. Figure 3 shows the use case contents. We detail and exemplify its elements in turn.

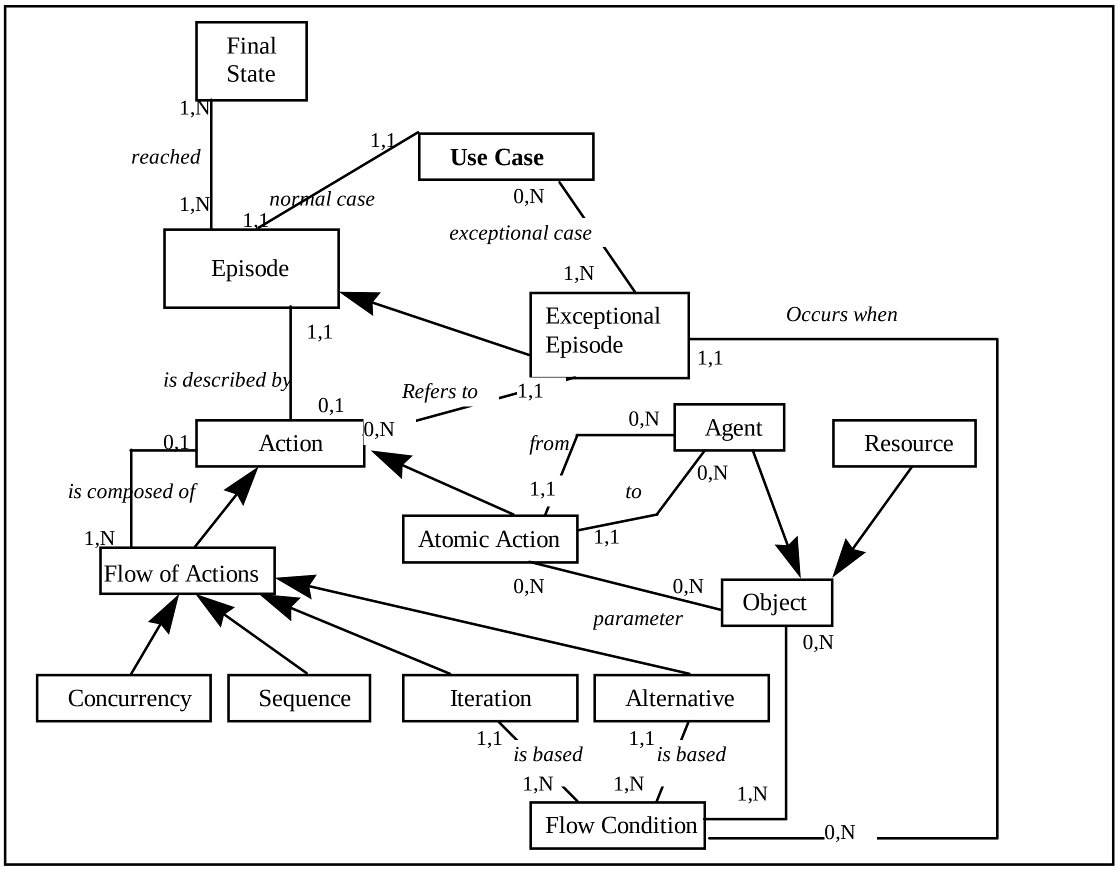


Figure 3 : Use case Contents.

The sentence *''the user inserts his card in the ATM''* is an example of *atomic action* from the *user* agent to the *ATM* agent. The parameter of this action is a resource (the '*card'*). Atomic actions are of two kinds : *communication actions* between two different agents and *internal actions* in which the *from* and the *to* agents are the same. The previous example of action is an illustration of a communication action whereas the sentence *''the ATM checks if the code is valid''* is an example of internal action. Moreover we distinguish four types of communication actions : *service request, service provision, information request* and *information provision*. In a service request action, an agent A asks a service to an agent B, whereas A proceeds to the satisfaction of the requested service in the case of a service provision action to B. *''The user inserts his card in the ATM''* is an example of service request action which is satisfied by the service provision action *''the ATM delivers the cash to the user''*. An information request is a communication action between agents for asking to be provided information. *''A prompt for code is given by the ATM to the user''* is an example of information request action which expects the performance of the information provision action *''the user inputs the code in the ATM''*.

A *flow of actions* is a complex action composed of other actions. The composition is based on the sequence, concurrency, iteration and alternative constructors. The sentence *''the user inserts the card in the ATM ; then, the ATM checks if the card is valid''* is an example of a flow of actions comprising two atomic actions (*insert* and *check)*. The flow of actions *''the ATM keeps the card, meanwhile the ATM displays an « invalid code » message to the user''* is an illustration of a *concurrency* based composition of actions in the flow. There is no specific order between the two actions.

An *iteration* can be expressed as follows : *''repeat less than four times and until the amount is valid : a prompt for amount is displayed by the ATM to the user, the user enters an amount in the ATM, the ATM checks if the amount is valid''*. In this example, the iteration encapsulates a sequence of three atomic actions, namely *display*, *enter* and *check*. The iteration condition is expressed by the clause *''less than four times and until the amount is valid''*. Finally *''if a receipt is asked by the user to the ATM, then the ATM prints a receipt to the user''* is an illustration of an *alternative flow of actions*. The condition is written *: ''if a receipt is asked by the user to the ATM''*.

Flow conditions are necessary to integrate several courses of actions in one complex flow of actions. In the last example, the flow of actions integrates the description of what happens when the condition : *''if a receipt is asked by the user to the ATM''* is true.

An *episode* comprises the flow of actions and the corresponding *final states*. There are several possible final states for an episode and therefore, the flow of actions can include several paths to reach each of the possible final states. An episode is *normal* when each of its possible final states ensures the fulfilment of the user goal else it is said *exceptional*. An *exceptional episode* describes a « non normal » course of actions reaching a final state which does not fulfil the goal of the initiating agent. In the ATM case study, the normal episode corresponds to the flow of actions ending into cash delivery to the user (with or without a receipt). *''If the code is not valid and the ATM keeps the card''* describes a non normal course of actions that will be part of an exceptional episode. An exceptional episode has an *occurrence condition* and includes a reference to an action of the normal episode in which the occurrence condition can become true.

The use case specification must provide a completed description of all the normal and exceptional flows of actions (see in the appendix the complete specification of the ATM withdraw cash use case). Because of its complexity, our approach proposes to construct it incrementally, taking into account partial stories called scenarios. In our view, a *scenario* is the description (possibly incomplete) of a flow of actions illustrating one path either in a normal or in a non-normal episode.

2.3 Relationship between natural language and use case model

The use case model provides the structure of the use case specification. The specification itself is written in natural language. Consequently, there is a relationship between the text structure and the use case structure which is highlighted in figure 4.

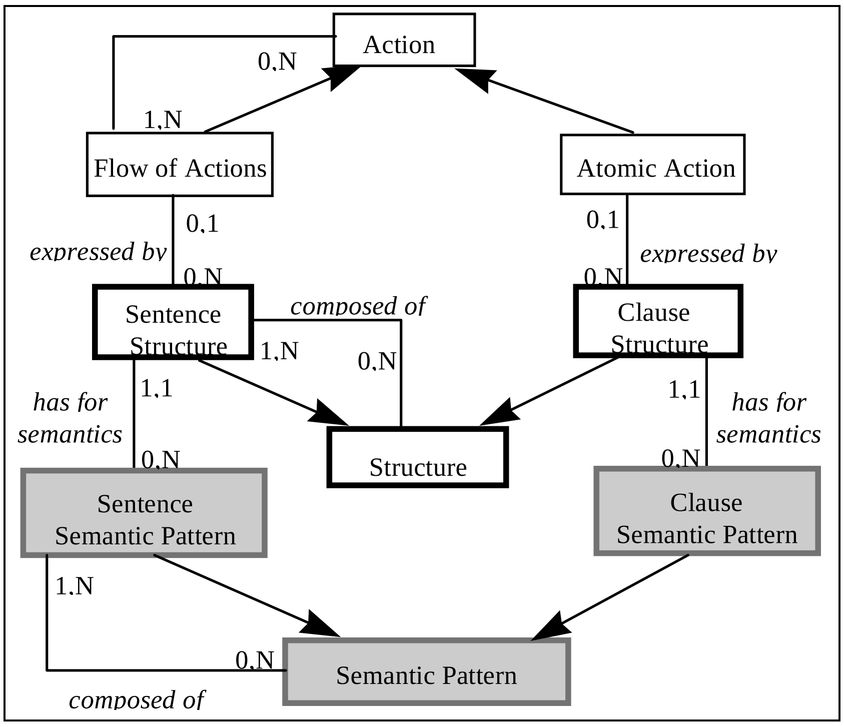


Figure 4 : Relationship between text structure and the action structure.

The text *structure* (see bold black boxes of figure 4) can be decomposed into more elementary structures which are either *clause structures* or *sentence structures*. For example, the text *''the user inserts the card in the ATM ; then, the ATM checks if the card is valid''* is a sentence structure decomposed into two elementary clause structures corresponding to the clauses : *''the user inserts the card in the ATM''*, and *''the ATM checks if the card is valid''*. The text below is a more complete example of the textual specification of the normal episode of the ATM example.

The **normal case** of the use case *Cash1* **named** NormalCase **is described by** :

The user inserts the user's card in the ATM. The ATM checks if the card is valid. If the card is valid repeat less than four times and until the code is valid ; a prompt for code is given by the ATM to the user, the user inputs the code in the ATM, and the ATM checks if the code is valid.

If the code is valid before the fourth time, repeat less than four times and until the amount is valid ; a prompt for amount is displayed by the ATM to the user, the user enters an amount in the ATM, the ATM checks if the amount is valid.

If the amount is valid before the fourth time then the ATM ejects the card to the user. The ATM asks the user if the user wants a receipt. The user enters if the user wants a receipt in the ATM.

If a receipt is asked by the user to the ATM then the ATM prints a receipt to the user.

Even if the receipt is not asked by the user, the ATM delivers the cash to the user.

This episode permits **to reach the following states** : the user has the card, the user has cash, sometimes the user has a receipt, the ATM is ready to serve.

Sentence and clause structures correspond to the surface structures of the textual specification. They have a meaning which is respectively provided by *sentence* and *clause patterns*. The deep structure of the use case specification is provided by the sentence and clause patterns (see *grey boxes in figure 4)*. Sentence patterns provide the semantics of sentences expressing sequence, conditional flows etc. Clause patterns give the semantics of actions such as service provision actions, information request actions, etc.

Sentence and clause patterns which are case patterns in a *case grammar* are presented in more detail in the following section.

3. Linguistic patterns and structures for use case specification

3.1 Introduction and state of the art

Our approach of use case specification is based on Natural Language (NL) text. There is thus, a necessity for catching the semantics of text in order to fill the gap between the informal representation and the formal model of use cases. The Case Grammar, unlike other representations of the semantics such as Sowa's graphs[2] [43], or Montague's grammar[3] [10] focuses on the notion of action which is central in our use case model. The case grammar, which intends to represent a « universal » semantics is a convenient way to do it, in the sense that it permits us to catch both the semantics of NL and the semantics of the use case model.

Fillmore [13] has first shown that, in a natural language sentence, the semantic relationship between the main verb and the other participants, is very different from the classical grammatical cases of flexional languages such as Latin, German or Russian. Fillmore distinguishes between the surface level cases corresponding to grammatical functions, and the deep level cases (called *semantic cases*) corresponding to underlying roles that the sentence participants play with respect to the main verb. Fillmore's case grammar is founded on a set of

[2] Sowa's conceptual graphs provide a meaning to concrete concepts by relating them to a semantic network in which abstract concepts represent a long-term domain or contextual knowledge.

[3] Montague's grammar identifies the logic of predicates as the semantics aspect of language. The meaning of a sentence is represented by a logic structure including quantifiers, logical connectors, etc.

semantic cases and a set of rules, which permit to provide the semantics of simple sentences describing actions.

Many other authors have followed the way opened by Fillmore, proposing many different definitions of the notion of semantic case. According to Schank [40], cases are used to define a small set of primitive actions. The semantic cases are thus reduced to a small set corresponding to the semantic functions of the elements involved in such actions. According to Wilks [46], semantic cases are elementary units of semantics used both to define the semantics of action verbs and sentences. According to Simmons [41], the semantic nature of the relationship between the verbal predicate of a sentence and the other participants is identified by a combination of semantic cases. Thus, unlike Fillmore's definition, a single case is only a part of the semantic relationship between the elements of the sentence and the action verb. In Boguraev and Spark-Jones [2] the choice of the list of cases relies on numerous experiments, in order to avoid controversy. A very interesting definition comes from Dik [9] who makes a clear distinction between *state of affairs* and *predications*. State of affairs are templates that represent the linguistic knowledge on the semantics of a surface level verbal predicate. Predications correspond to the semantic structure underlying an analysed sentence. The typology of state of affairs is based on a semantic qualification of verbs. Unlike Fillmore, Dik considers that such a typology justifies the choice between possible cases during the analysis of a sentence by a predication.

Another interesting difference lies in the conceptualisation of the knowledge about sentences. According to Schank, the elements that appear in the deep level structure may not appear at the surface level. Such information can come from a contextual or domain knowledge of the verb, or from interactions with a user. For example, Schank's representation of *''the user reads the prompt''* includes the user's eyes so as to support the realisation of the action. To eliminate this kind of consideration, Dik distinguishes between specific knowledge on the semantic of a verb and more contextual information on the semantic structure of a sentence. Thus, the semantic definition of a verb does only include the set of obligatory arguments that appears at the surface level.

Numerous adaptations of the case grammar were proposed since the first version of Fillmore’s case grammar. Our choice focuses on the necessity of analysing the actions of use case specifications, as defined in the previous section. In this view, we found that Schank’s strategy is not adapted. Indeed, the applications specified in use cases can belong to a very high number of various domains. Thus, the cost of associating domain or contextual knowledge to verbs is too high compared to the contribution. Similarly to Simmons, we do not put exclusivity constraints on cases at the moment of analysis (e.g. the user is both the initiator of the action and the source of the information in the sentence *''the user enters his code in the ATM''*). In our view, the issue solved by Wilks’ semantic units is of a pure linguistic value. Indeed, use case analysis requires only to delineate the actions and the object that participate to these actions within the sentences of a use case. Using a unified view of the semantics of objects, verbs and sentences does not contribute in the analysis result.

Based on this argumentation, our work assumptions for defining a case grammar are :

1. To differentiate, as Dik does, a generic knowledge of actions, from the use of this knowledge at the level of complex sentences. We call this generic knowledge *semantic patterns*.

2. To justify our choice of semantic patterns by the semantics of the use case model. Indeed, the semantic field covered by semantic patterns is limited to the one of the actions of the use case model. Similarly to Dik, it is the choice of our patterns that justifies our list of cases.
3. As Dik and Simmons do, we allow the use of several cases to define the role of one element in a sentence.
4. In opposition to Schank, we do not include domain or context knowledge in semantic patterns. As Dik's, our approach is domain independent.

As suggested in this list of assumptions, our case grammar is not universal. We have already applied our list of patterns on a set of fifteen real case studies written by industrials, and found out that they do not apply to about 10 % of the contents of the use cases (e.g. adverbs, negation and numeration). We are currently working on how to reduce this proportion, and we are searching for the significance, at the level of the RE process, of having a text element not analysed by our patterns.

3.2. Semantic patterns and use case model

Our case grammar introduces a set of semantic cases such as agent, object, destination, etc. Semantic cases define the semantic roles that the different elements of an action clause play with respect to the main verb. The semantic cases of the case grammar are different from grammatical functions. In the clause *''the card is ejected''* the grammatical subject (*'the card'*) is the object of the action whereas in *''the system ejects the card''*, the subject (*'the system'*) is the agent of the action. The semantic cases *agent* and *object* are thus different from the grammatical functions subject and direct object. The differentiation between semantic cases and grammatical functions respects Chomsky's well known hypothesis [5] following which, a deep structure of the language, the semantics, must be differentiated from a surface structure of language, the grammar. Indeed the purpose of the semantic cases is to remove the ambiguity introduced by grammatical functions. In the case grammar, the agent is the entity which undertakes or controls the action, and the object is the entity which undergoes the action. These definitions are orthogonal to the language grammar and syntax; indeed the agent and object cases can as well appear as subject or complement in a clause.

An important result of the Case Grammar is that verbs which have a similar meaning are used with similar semantic structures (state of affairs in Dik's terminology). For example, the verbs expressing an action of communication (such as "to give', "to take", "to move", "to enter", etc.) always express the move of an object (information or physical object), initiated by a agent, from a source location to a destination location. Typical semantic structures are defined by patterns of cases called *semantic patterns*. Semantic patterns are represented as templates that permit to associate a semantic case to the different elements of clauses and sentences. In our case, the purpose of the semantic patterns is to define the semantics of the clauses and of the sentences which are respectively expressed in the atomic actions and the flows of actions of use case specifications. Thus, our ontology of semantic patterns presented in figure 5 is fully directed towards the semantics definition of the different types of actions included in the use case model.

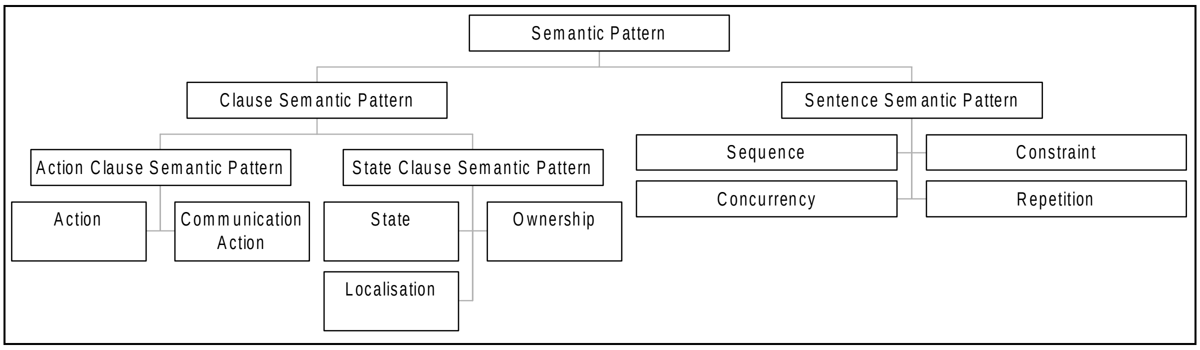


Figure 5: Ontology of semantic patterns.

This figure shows that our case grammar comprises semantic patterns for sentences and clauses. At the level of clauses, case patterns are *clause semantic patterns* associated to verbs. Similarly to Dik's state of affairs, clause patterns provide the intrinsic semantics of verbs, and do not include any domain nor context knowledge such as in Schank's grammar. Clause patterns can define the semantics of actions (e.g. *communication actions*) and static relationships (e.g. *state*, *localisation*, or *ownership*).

Action clause semantic patterns provide the semantics of the atomic actions of the use case model by associating a semantic role to the related agents and parameter objects.

State clause semantic patterns provide the semantics of initial and final states of objects. The ownership and localisation define the respective roles in a relationship between two objects, whereas the state pattern identifies the specific static properties of one object. In addition, for the initial and final states of objects, the state clause semantic patterns define the semantics of object states on which rely the flow conditions of flow of actions. Clause semantic patterns are presented in the form N (V) [C], where N is the name of the pattern qualifying the intrinsic semantics of the verb V, and C is the list of cases to be associated to the elements of the analysed clause. To represent the semantics of a clause in a use case specification consists in instantiating a clause semantic pattern, that is to say in giving a value to the variables V and C. For example,

Communication (V) [Agent; Object; Source; Destination]

being the clause pattern for communication actions, the instantiated clause pattern :

Communication (‘insert’) [Agent : ‘the user’; Object : ‘his card’; Source : ‘the user’; Destination : ‘the ATM’]

provides the semantics of the atomic action *‘’the user inserts his card in the ATM’’* by providing a role to the agents and object parameters of the action.

In addition to clause semantic patterns, we use sentence semantic patterns to define, as illustrated in figure 4, the semantics of the flows of actions of the use case specification. Sentence semantic patterns, contrarily to clause patterns define the semantic relationships between several clauses or sentences of a same text. This recursive property of sentence semantic patterns reflects to the composition of flows of actions by several actions in the use case model. For example :

Sequence [Before : Communication (‘insert’) [Agent : ‘the user’; Object : ‘the card’; Source : ‘the user’; Destination : ‘the ATM’] ;

After : Sequence[Before : Communication ('display') [Agent : 'the ATM' ; Object : 'a welcome message' ; Source : 'the ATM' ; Destination : 'the user'];
After Action ('check')[Agent : 'the ATM' ; Object : 'the card']]]

gives the semantic of the sequential flow of three atomic actions *''The user inserts the card in the ATM. Then, the ATM checks the card, but before, the ATM displays a welcome message to the user''*.

Similarly, the sentence semantic pattern of constraint relates a condition to any other constrained sentence or clause. The semantics of the condition and of the constrained sentence can be itself analysed through semantic patterns which will respectively define the semantics of the flow condition related to the flow of actions, and of the constrained sentence.

The same relationship between sentence semantic patterns and flows of actions of the use case model applies to the concurrency and repetition patterns. The concurrency sentence semantic pattern provides the semantic relationship of co-occurrence of several actions, like in *''while the user enters the code in the ATM, the ATM checks the receipt paper''*. The repetition pattern identifies the repetition of actions under a certain condition, such as in *''the user enters the code in the ATM until it is valid''*.

3.3. Semantic patterns and surface structures

An important property of semantic patterns is univocity. Indeed, semantic patterns give a unique deep level representation to support the meaning of a chunk of text. On the contrary, natural language often provides many alternative ways to express the same knowledge.

However restricted by syntactic rules, the sequencing of words at the surface level of a sentence may vary without affecting the deep meaning of that sentence. We call *structure* the different surface representations of a semantic pattern.

Consider for example the action clause semantic pattern : Action (V) [Agent; Object], where the agent is the active entity that initiates or controls the action, and the object is the passive entity which undergoes or results of the action. The eleven structures, described below S1 to S11, identify eleven surface ways to express actions. Either through verb groups which main verb is a verb of action (such as to create, to modify, to check, to erase, or to agree), or through noun groups which main noun corresponds to the substantive form of verbs of action (the creation, the modification, etc).

S1 : [[NG](Subject)$_{\text{Agent}}$ [Verb](Main Verb)$_{\text{Action}}$ [NG](Complement) $_{\text{Object}}$](VG active)$_{\text{Action}}$

S2 : [[NG](Subject)$_{\text{Agent}}$ [Verb](Main Verb)$_{\text{Action}}$] (VG active)$_{\text{Action}}$

S3 : [[NG](Subject)$_{\text{Object}}$ [Verb] (Main Verb)$_{\text{Action}}$] (VG active)$_{\text{Action}}$

S4 : [[NG](Subject)$_{\text{Agent + Object}}$ [Verb](Main Verb)$_{\text{Action}}$](VG active) $_{\text{Action}}$

S5 : [[Verb](Main Verb)$_{\text{Action}}$ [NG](Complement) $_{\text{Object}}$](VG active)$_{\text{Action}}$

S6 : [[NG](Subject)$_{\text{Object}}$ [be Verb](Main Verb)$_{\text{Action}}$](VG passive)$_{\text{Action}}$

S7 : [[NG](Subject)$_{\text{Object}}$ [be Verb](Main Verb)$_{\text{Action}}$](VG passive)$_{\text{Action}}$

S8 : [[N substantive form of action Verb](Main Noun)$_{\text{Action}}$ [of NG](Epithet)$_{\text{Agent}}$](NG)$_{\text{Action}}$

S9 : [[N substantive form of action Verb](Main Noun)$_{\text{Action}}$ [of NG](Epithet)$_{\text{Object}}$ [by NG] (Epithet)$_{\text{Agent}}$](NG)$_{\text{Action}}$

S10 : [[NG] (Main Noun)$_{\text{Agent}}$[N substantive form of action Verb](Noun)$_{\text{Action}}$ [of NG](Epithet)$_{\text{Object}}$](NG)$_{\text{Action}}$

S11 : [[N](Main Noun)$_{\text{Object}}$ [Verb passed participle](Epithet)$_{\text{Action}}$](NG)$_{\text{Action}}$

For example, the atomic action *‘’the ATM checks the card’’* (here expressed using structure S1) can also have the surface representations :

(S1) *‘’the ATM is checking the card’’*,

(S7) *‘’the card is checked by the ATM’’*,

(S9) *‘’the checking of the card by the ATM’’*,

(S10) *‘’the ATM checking of the card’’*.

Such structure naturally occurs in sentences such as : *‘’the checking of the card by the ATM occurs before the code prompt display’’*.

Similarly, to each clause semantic pattern and to each sentence semantic pattern is associated a list of surface structures which provide numerous ways of expressing atomic actions, flows of actions and states in the use case specification. As sentence patterns and flows of actions, sentence structures have a recursive definition which provides more freedom of expression.

3.4. Structures, patterns, and use case model

Let us now summarise the connection between structures, semantic patterns, and the use case concepts. Each surface structure instantiates a semantic pattern which gives the semantics of the corresponding chunk of text. Clause structures permit to identify clause semantic patterns, and sentence structures permit to identify sentence semantic patterns. As said earlier, each semantic pattern can be expressed by several semantic structures.

On the other side, each semantic pattern identifies the semantics of the use case concepts. For example the communication action clause semantic pattern corresponds to an atomic action with its name, agents and parameters in the use case model. Similarly, the action clause pattern identifies atomic actions. The sentence patterns of sequence, constraint, concurrency and repetition permit to identify, respectively, the flows of actions of the use case model : sequence, concurrency, iteration and alternative. The state clause semantic patterns identify object states, the flow conditions of iterations and alternatives, and occurrence conditions.

Complementarily, the semantics of each element of the use case model is expressed by semantic patterns : the atomic actions by action clause semantic patterns, the flows of actions by sentence semantic patterns, the flow conditions and the states by state clause semantic patterns. The semantic patterns being expressed using certain surface structures.

As an example, the following chunk of use case specification :

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

corresponds to the pattern instance :

Communication ('insert') [Agent : 'a user' ; Object : 'a card' ; Source : 'the user' ; Destination : 'the ATM]

This communication action clause semantic pattern can be expressed using the structure :

[['A card'](Subject)$_{Object}$ *['is inserted'](Main Verb)*$_{Communication}$ *['by a user'](Complement)*$_{Agent+Source}$ *['into the ATM'](Complement)*$_{Destination}$*](VG active)*$_{Communication}$

which corresponds to :

''A card is inserted by a user into the ATM''

Identifying the concepts of the use case model from natural language is a two stage process which requires first the semantic analysis of the text and then the mapping of the resulting semantic patterns onto the concepts of the use case model. These two stages are illustrated in section 5 which describes more extensively the process of constructing the use case specification of the ATM example.

4. Overview of the guided process

The output of the use case specification process is a text structured according to the use case model and the linguistic templates presented in section 3. Our assumption is twofold : (1) due to its complexity a use case cannot be produced in one shot and, (2) forcing the author to write the sentences according to hard coded patterns [6, 24] is far too constraining. Thus, we opted for front end descriptions which are partial views of the course of actions in natural language prose.

As a consequence the process of use case specification is a stepwise process which guides the progressive transformation of input prose texts (the scenarios) into structured texts and their integration in the use case specification. The process comprises four main activities to :

1. situate the use case,
2. capture the initial scenario and guide its completion if necessary,
3. integrate the scenario into the use case specification and,
4. infer the need for new scenarios and proceed to their capture and integration.

Activity 1 aims at capturing information about the context of the use case, including the use case name, the initiating agent, his/her goal, the service it describes and the initial state.

Activity 2 is an iterative one. It consists in : capturing the initial scenario, checking its completeness and ensuring its completion. The initial scenario is a narrative course of actions that can be incomplete and ambiguous. The final scenario description obtained at the end of activity 2 is a complete description of a pathway in an episode expressed unambiguously according to our semantic patterns (see section 2). The initial text is transformed in order to remove ambiguities of natural expressions and completed to match the definition of an episode pathway.

Activity 3 integrates the scenario into the episode structure of the use case. The output scenario of activity 2 corresponds to one path in an episode in the graph of episodes of the use case. The step performs the necessary actions to infer from flow conditions the right positioning of the scenario in the use case.

The reasoning underlying activities 2 and 3 may lead to the emergence of new scenarios. Selecting one of those triggers a new sequence of activities 2, 3 and 4 in the process. In fact, the process is an iterative process where the four mentioned activities are tightly interleaved.

Each activity is supported by *guidelines* to help the author to perform the requested actions and *rules* to map the input prose's onto the internal use case structure and reason about it. The overall architecture of the process is sketched in figure 6. The front end is a guided communication with the user. The back end is based on rules working on the use case structure.

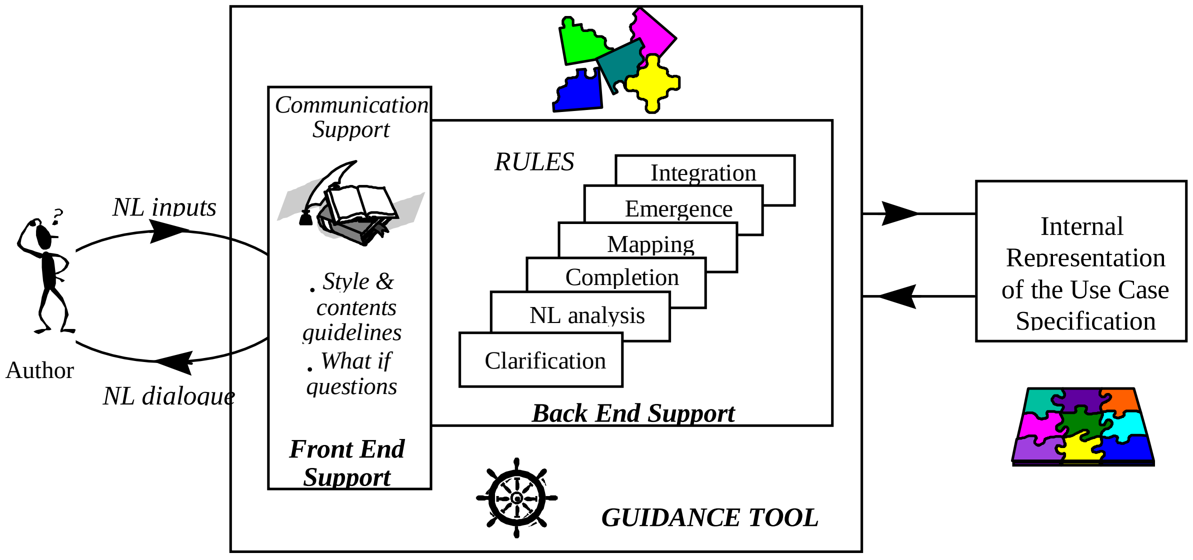


Figure 6 : Overall architecture of the process.

Guidelines are of two types : *style guidelines* provide recommendation on the style of writing narrative prose; *contents guidelines* advise the author on the expected contents of his/her prose. Rules are of five types : (1) *analysis rules* analyze the semantic contents of each sentence against the linguistic patterns; (2) *mapping rules* map the elements of instantiated patterns into elements of the use case model, (3) *refinement rules* include the clarification and completion (3) *integration rules* and (4) *emergence rules*. The remaining of this section illustrates guidelines and rules, respectively. More will be described in the next section with a walk through the ATM example guided process.

Style and contents guidelines have the form of plain text which can be prompted to the use case author on demand while writing down a scenario. Some examples are shown below.

Style guidelines :

1- You should avoid the use of anaphoric references such as « he », « she », « it » « his »or « him ». Instead, you should use nouns defined in the use case glossary.

2- You should avoid the use of synonyms and homonyms. The same object or agent should be named identically through out all texts.

3- You should avoid sentences with more than two clauses, each clause being composed of a subject, a verb and its complements.

4- You should mention explicitly your assumption when some action is done under certain conditions. For example you can use ‘’If <condition> then Action’’.

5- You should mention explicitly the condition for stopping repeated actions. For example you can use ‘’Repeat <action> until <condition>’’.

6- You should mention explicitly the co-occurrence of several actions. For example, you can use ‘’While <action>, <co-occurring action>’’.

7- When you express an action, you should mention explicitly the agent which undertakes the action, and the object of the action. Thus prefer the active to the passive voice.

8- When you express an action of communication, you should mention in addition the source and the destination of the communicated object.

9- You should write sentences at the present tense, avoid the use of the negation, of adverbs and of modal verbs.

Contents guidelines :

The expected scenario prose is a description of a single course of actions. Alternative scenarios, interruptions or exceptional treatments are described separately. A course of actions typically describes sequentially ordered actions : actions from an agent to the system, system responses to the agent, communications between agents and possibly actions internal to the system. You should put your sentences in the order of the scenario history. The course of actions should be completed with the resulting end states. You should describe the course of actions you expect, not the actions which are not expected.

Below, we exemplify the different types of *rules* mentioned above. All rules have a premise and a body separated by a «→ ». The premise defines the precondition for executing the body. It is described by a first order logical formula. This precondition is expressed on elements of the use case specification and defines the situation of the product under construction in which the rule is applicable. The body is either an action required from the author (using the ASK predicate) or the generation of new elements of the use case specification (using the GENERATE predicate). In the prototype tool under development, rules are implemented as PROLOG clauses. The enactment mechanism of the guided process is therefore, built on top of the PROLOG inference engine.

Analysis rules :

The following rule aims at generating the action pattern from a clause having a subject and a verb expressed in the active form.

AN1 : $\forall$ *NG, V :*
[[NG](Subject)$_{Object}$ *[V] (Main Verb)*$_{Action}$ *] (VG active)*$_{Action}$ $\rightarrow$
GENERATE(Action(V)[Agent:? ; Object:NG])

Similarly, AN10 instantiates the communication action pattern from a clause structure including a subject, a communication verb at the active voice, an object, and a complement.

AN10 : $\forall$ *NG1, NG2, NG3, V :*
[[NG1](Subject)$_{Agent+Source}$ *[V] (Main Verb)*$_{Communication}$ *[NG2](Object)*$_{Object}$ *[to NG3](Complement)*$_{Destination}$*] (VG active)*$_{Action}$ $\rightarrow$
GENERATE(Communication(V)[Agent:NG1 ; Object:NG2 ; Source:NG1 ; Destination:NG3])

Clarification rules.
CL1 expresses that if an anaphoric reference is employed in a sentence, then, the use case author is asked to replace it by a noun from the glossary.

CL1 : $\forall$ *A :*
(Action [Agent:A ; Object: _] $\vee$
Action [Agent: _ ; Object:A] $\vee$
Communication [Agent: _ ; Object: _ ; Source:A ; Destination: _] $\vee$
Communication [Agent: _ ;Object: _ ; Source: _ ; Destination:A] $\vee$
State [Object: _ ; State:A] $\vee$ *Ownership [Owner:A ; Owned: _]* $\vee$
Ownership [Owner: _ ; Owned:A] $\vee$ *Localisation [Location:A])* $\wedge$
Anaphoric Reference (A) $\rightarrow$
ASK(« Clarify A by replacing this anaphoric reference by a noun defined in the glossary »)

Note : « _ » is used to denote an anonymous variable which value is of no importance. The predicate 'Anaphoric Reference (A)' identifies if the term A includes a pronoun (he, his, him, etc.).

Rule CL7 permits to clarify the type of a condition by asking the use case author. More precisely, if a condition is mentioned in the prose text, then ask for confirmation : is it an initial state condition (true all along the use case) or a flow condition which influences the flow of actions in the use case?

CL7 : $\forall$ *C, D :*
Constraint [Condition:C, Constrained:D] $\wedge$
Conditional (Is Based on Flow Condition : C, Composed of : D)$\rightarrow$
ASK(« What is the type of the condition C ? (initial state or flow condition) »)

Completion rules :

Among all completion rules, we defined seven completion rules which aim at completing pattern not fully instantiated. Therefore, such rules express that if the instantiated pattern of a clause has a missing element then the author is asked to provide the missing element. The example below applies when an action does not have an agent.

CO1 : ∀ V , ∃ O:

Action (V) [Agent:? ; Object:O] →

ASK(« Complete : V by... (agent of the action) »)

Another set of completion rules are defined for checking the completion of a service request and its service provision as well as an information request and its provision. For instance, if there is no « service request » action to an action of the type « service provision » from agent A to system B then, the missing action should be questioned.

C08 : ∀ V1, S, A, B :

Atomic Action(Name : V1, From Agent : B, To Agent : A, Parameter : S) ∧

(Type(V1) = 'Service Provision')) ∧

¬(∃ V2 : Atomic Action (Name : V2, From Agent : A, To Agent : B, Parameter : S)) ∧

(Type(V2) = 'Service Request') ∧ Follow(V1, V2) →

ASK(« Complete service provision V1 with the anterior action :... (request of service S from A to B))»

Note : Follow(V2, V1) is a predicate which value is true when V2 appears after V1 in the flow of actions, and false otherwise. Type(V) is a function returning the type of the action, such as service provision, service request, etc.

Mapping rules :

Rule MA1 establishes that if in the communication pattern the initiating agent is either the source or the destination of the communication then the corresponding atomic action must be generated in the use case specification.

MA1 : ∀ V, A, O, S, D :

Communication (V) [Agent:A ; Object:O ; Source:S ; Destination:D] ∧

(Unify (A, S) ∨ Unify (A, D)) →

GENERATE(Atomic Action (Name : V, From Agent : S, To Agent : D, Parameter : O))

Rule MA6 permits to generate an alternative flow of actions based on a flow condition, out of the constraint pattern.

MA6 : ∀ C, D :

Constraint [Condition:C ; Constrained:D] →

GENERATE(Alternative (Based on Flow Condition : C, Composed of : D))

Integration rules :

The rule IN2 supports the integration of an exceptional episode into a use case specification.

IN2 : ∀S, ∃U, A,C,M, F :

EpisodeStructure (Episode : S, Use Case : U, Action : A) ∧

CaseToConsider (Episode : S, Use Case : U ,Type : ‘Exceptional‘, FlowCondition :C ,_ , EpisodeName : M) ∧ FinalState(Episode : S, Use Case U, Final State : F) →

ASK(« Provide the name N of the exceptional episode », N) ∧

GENERATE(ExceptionalEpisode(EpisodeName : N, Use Case : U, Condition : C, ReferredAction :RetrieveAlternative (Condition : C, Use Case : U, EpisodeName : M) , ReferredEpisodeName : M, Action : CopyThen (Action : A, Condition :C))) ∧

(∀F1 FinalState(Episode : S, Use Case U, Final State : F1) →

GENERATE(FinalState(EpisodeName : N, Use Case U, Final State : F1)))

Note : the RetrieveAlternative (C, U,M) function permits to get the identifier of the alternative embedding the flow condition C into the episode M of the use case U. The CopyThen (A,C) function returns the action mentioned in the "then" part of the alternative embedding the flow condition C in the flow of actions A. The CaseToConsider () predicate permits to reach all the cases generated by the rule EM1.

This rule supports the generation of the exceptional episode comprising *its name, the use case it belongs to, its occurrence condition,* the reference of the *action* in the normal *episode* where the *exception* happens, *its action* and also *all the reachable states.*

Emergence rules

EM1 : ∀N, ∃U,C :

NumberOfFC (ActionEpisode (EpisodeName : N, Use Case : U),n) ∧ (∀k ∈ {0.. n-1})→

ASK(«type of the Case $((\wedge_{j=1..k} C_j) \wedge \neg C_{k+1})$ *» ,type) ∧*

GENERATE(CaseToConsider(_ , Use Case : U ,Type : type, FlowCondition :C ,Case : $((\wedge_{j=1..k} C_j) \wedge \neg C_{k+1})$ *, EpisodeName : N))*

Em1 identifies all the cases which should be investigated by the use case author in order to complete the initial episode. It assumes that all the flow conditions are labeled with a particular number according to their level of nesting. For each case detected by the rule, the author is asked to classify it as normal or exceptional.

The application of rules and guidelines is exemplified in the next section with the ATM case study.

5. Walk through the process with the ATM example

This section is a walk through the guided process of use case specification with the ATM example. It is a use scenario illustrating how the use case author interacts with the tool providing guidance. Due to space limitation we concentrate on *one* use case specification (‘’withdraw cash’’). The presentation is organised in five steps which show the progressive transformation, completion and integration of input scenarios into the ‘’withdraw cash’’ use case specification.

Let us assume that the use case has been situated and its contextual information provided and stored in the use case structure (see in appendix the corresponding specification). The first step that we consider is therefore, the capture of the initial textual scenario which is intended to describe the normal course of actions.

5.1. Capturing the initial scenario

Let us call Mr. Bean the use case author. In order to help Mr. Bean the *style* and *contents* guidelines associated to the initial scenario writing are prompted on demand (see section 4). But Mr. Bean is aware of them and writes down his view of the normal course of interactions between a user who wants to get cash and the ATM as follows :

> The user inserts his card in the ATM. The ATM checks if the card is valid. Then, a prompt for code is given and the user inputs the code. If the code is valid, a prompt for amount is displayed. The user enters an amount. If the amount is valid, the ATM delivers the cash, but before the ATM ejects the card and a receipt is printed if this one was asked.
>
> *The final states are :* The user has the cash his card, and a receipt. The ATM is ready to serve.

The text is then scanned with the analysis rules and mapped onto elements of the use case specification using mapping rules.

5.2. Performing linguistic analysis and mapping on episode structure

As part of the linguistic approach described in section 3, *analysis rules* are used to identify the text structure, and to map the text onto instances of sentence and clause semantic patterns. This results in our example in the following set of instantiated patterns :

```
Communication (insert) [Agent:‘the user’ ; Object:‘his card’ ; Source:‘the user’ ; Destination:‘in the ATM’]
Sequence [Before : Action (check) [Agent:‘the ATM’ ; Object:State [Object:‘the card’ ; State:‘valid’]] ;
     After : Sequence [Before : Communication (give) [Agent:? ; Object:‘a prompt for code’ ; Source:? ;
     Destination:?] ;
             After : Communication (input) [Agent:‘the user’ ; Object:‘the code’ ; Source:‘the user’ ;
             Destination:?]]
Constraint [ Condition : State [Object:‘the code’ ; State:‘valid’] ;
     Constrained : Communication (display) [Agent:? ; Object:‘a prompt for amount’ ; Source:? ;
     Destination:?]]
Communication (enter) [Agent:‘the user’ ; Object:’an amount’ ; Source:’the user’ ; Destination: ?]
Constraint [Condition : State [Object:‘the amount’ ; State:‘valid’] ;
     Constrained : Sequence [Before : Sequence [Before : Communication (eject) [Agent:‘the ATM’ ;
                               Object:‘the card’, Source:‘the ATM’ ; Destination:?] ;
                           After : Constraint [Condition : Communication (ask) [Agent:? ;
                                       Object:‘this one’ ; Source:? ;
                                       Destination:?];
                                 Constrained : Communication (print) [Agent:? ;
                                        Object:‘a receipt’; Source:?;
                                        Destination:?]]]
                After : Communication (deliver) [Agent:‘the ATM’ ; Object:‘the cash’ ;
                   Source:‘the ATM’ ; Destination:?]]]
The final states are :
Ownership [Owner:‘the user’ ; Owned:‘the cash’]
Ownership [Owner:‘the user’ ; Owned:‘his card’]
Ownership [Owner:‘the user’ ; Owned:‘a receipt’]
State [Object:‘the ATM’ ; State:‘ready to serve’]
```

Table 1 : Instantiated patterns.

The analysis of Mr. Bean’s initial scenario leads to the instantiation of ten action clause patterns within which nine are communication action clause patterns. The instantiation of the action and communication action clause patterns from the input text provides values to the agent, source, object and destination cases. Question marks characterise missing elements in the input text ; they will be used for completion. The analysis rules identify the agents of the scenario out of the agent, source and destination cases : *‘the user’* and *‘the ATM’*. Then, the object case identifies the resources used in each atomic action of the flow of actions *: ‘card’, ‘prompt for code’, ‘code’, ‘prompt for amount’, ‘amount’, ‘receipt’*, and *‘cash’*. Action names, agents and objects identified by the analysis rules are used to fill in the glossary attached to the use case family.

Moreover, the analysis of the initial scenario shows that the actions are related through four sequence sentence patterns and three constraint sentence patterns. When the sequencing of actions is not respected in the initial scenario, such as in *‘’…the ATM delivers the cash, but before, the ATM ejects the card…’’*, the sequence patterns instances identify the real ordering.

Based on these pattern instances, *mapping rules* are automatically used to produce a natural language specification of the flow of actions the normal episode which has been just identified. For sake of clarity we use an indented presentation of the flow of actions, and we associate a unique identification number to each action. The plain text, as given above is equivalent in contents and meaning.

1. The user inserts his card in the ATM.
2. The ATM checks if the card is valid.
3. If the card is valid
    4. Then
    5. A prompt for code is given.
    6. The user inputs the code.
    7. If the code is valid
        8. Then
        9. A prompt for amount is displayed.
        10. The user enters an amount.
        11. If the amount is valid
            12. Then
            13. The ATM ejects the card.
            14. If this one was asked.
                15. Then
                16. A receipt is printed.
                17. The ATM delivers the cash.

Final states : The ATM is ready to serve. The user has cash. The user has his card. The user has a receipt.

Table 2 : Flow of actions after analysis and mapping of the initial scenario.

Let us comment this mapping. First, based on the action and communication action clause patterns instantiated during the initial text analysis, atomic actions are identified through rules MA1, MA2 and MA3. The terminological consistency of atomic action names is ensured at the analysis stage, through the rewriting of action verbs at the infinitive in pattern instances.

MA1 : ∀ V, A, O, S, D : Communication (V) [Agent:A ; Object:O ; Source:S ; Destination:D] ∧ (Unify (A, S) ∨ Unify (A, D)) → GENERATE(Atomic Action (Name : V, From Agent : S, To Agent : D, Parameter : O))

MA2 : ∀ V, A, ∃ O : Action (V) [Agent:A ; Object:O] ∧ Agent (O) → GENERATE(Atomic Action (Name : V, From Agent : A, To Agent : O))

MA3 : ∀ V, A, ¬∃ O : Action (V) [Agent:A ; Object:O] ∧ Agent (O) → GENERATE(Atomic Action (Name : V, From Agent : A, To Agent : A, Parameter : O))

As stated in these rules, the atomic actions are analysed separately, even if they occur in the same sentence. Communication action pattern instances lead to the mapping of an atomic action. As our purpose in not to rephrase Mr. Bean's scenario, the expression of the atomic actions identified in the initial text is not modified.

MA5 : ∀B, A : Sequence [Before:B, After:A] → GENERATE(Sequence (Composed of : B, A))

MA6 : ∀ C, D : Constraint [Condition:C, Constrained:D] → GENERATE(Alternative (Based on Flow Condition : C, Composed of : D))

In addition, based on the sequence patterns, atomic actions have been organised in the right sequence owing to rule MA5. For example, the sentence *''the ATM delivers the cash, but before, the ATM ejects the card, and a receipt is printed if this one was asked''* has been split into *''(13) The ATM ejects the car. (14) If this one was asked (15) Then (16) A receipt is printed (17) The ATM delivers the cash''*. When no explicit sequencing of the actions is expressed, the ordering of the sentences in the initial scenario is respected.

Then, flow conditions such as *''if the card is valid''* or *''if the code is valid''*, are identified from constraint pattern instances through rule MA6. Once identified, the alternative flows of actions are isolated, and the corresponding flow conditions are put in the order provided by the constraint pattern instances. For example the sentence *''If the code is valid, a prompt for amount is displayed''* becomes *''(7) If the code is valid (8) Then (9) A prompt for amount is displayed''*.

5.3. Linguistic clarification and completion

The linguistic analysis provides a baseline for linguistic based clarification and completion of the identified atomic actions. Both rules rely on situations identifying possible linguistic ambiguities in the expression of the atomic actions.

The *clarification rules* are used to change the wording and to remove possible ambiguities. Even if the first style guideline recommends to ''avoid the use of anaphoric references such as he, she, it, his and him'', it is necessary to check systematically the text provided by the author. The grammatical analysis performed as a pre-requisite for analysis rules provides the information required for these checks. Clarification rule CL1 uses this information and proposes to replace anaphoric references by nouns of the use case glossary (the underscore '_' is used for anonymous variables whose value is of no importance).

CL1 : ∀A : (Action [Agent:A ; Object: _] ∨ Action [Agent: _ ; Object:A] ∨
Communication [Agent: _ ; Object: _ ; Source:A ; Destination: _] ∨
Communication [Agent: _ ;Object: _ ; Source: _ ; Destination:A] ∨ State [Object: _ ; State:A] ∨
Ownership [Owner:A ; Owned: _] ∨ Ownership [Owner: _ ; Owned:A] ∨ Localisation [Location:A]) ∧
Anaphoric Reference (A) →
ASK(« Clarify A by replacing this anaphoric reference by a noun defined in the glossary »)

Mr. Bean has been using the term *'this one''* in the last sentence of his scenario. The clarification rule suggests to ''Clarify *this one* by replacing this anaphoric reference by a noun defined in the glossary''. Taking this suggestion into account, Mr. Bean now decides to modify the flow condition 14 and replace *'this one'* by *'a receipt'* which is a more explicit resource name that was inserted to the glossary during the analysis stage. The same applies to the first sentence and to the final state where Mr. Bean replaces the pronoun *'his'* by *'the user's'*. The action (1) thus becomes *''the user inserts the user's card in the ATM''*, and the final *state ''The user has the user's card''*.

As explained in the previous section, the instantiated patterns may highlight missing parameters through question marks. The c*ompletion* rules CO1 to CO7 help avoiding this form of incompleteness. In the ATM example, several communication action pattern instances are in the situation of one or several missing elements. For example, as shown in the analysis section, analysing the atomic action *''a prompt for the code is given''* instantiates the pattern *Communication (give) [Agent:? ; Object:'a prompt for the code' ; Source:?; Destination:?]* where the agent of the action, the source and destination of the communicated information are missing.

CO5 : ∀ V , ∃ O: Communication (V) [Agent : ? ; Object : O, Source : ? ; Destination : ?] / Atomic Action (V,_) → ASK(« Complete : V by ... (agent of the communication) from... (source of the communication) to... (destination of the communication) »)

Applying the completion rule CO5 displays the following template and asks Mr. Bean to fill it in : ''A prompt for the code is given *by... (agent initiating the communication) from... (source of the communication) to... (destination of the communication)*''. Making use of the template, Mr. Bean completes the sentence which becomes *''A prompt for the code is given by the ATM to the user''* in which the ATM is the new agent and source, and the user the new destination.

The systematic application of linguistic completion rules leads to the completion of eight actions. The new version of the current flow of actions specification is shown in table 3 where the supplementary elements are in bold, and the elements that have been clarified in bold and italic.

1. The user inserts ***the user's*** card in the ATM.
2. The ATM checks if the card is valid.
3. If the card is valid
    4. Then
    5. A prompt for code is given **by the ATM to the user**.
    6. The user inputs the code **in the ATM**.
    7. If the code is valid
        8. Then
        9. A prompt for amount is displayed **by the ATM to the user**.
        10. The user enters an amount **in the ATM**.
        11. If the amount is valid
            12. Then
            13. The ATM ejects the card **to the user**.
            14. If ***a receipt*** was asked **by the user to the ATM**
                15. Then
                16. A receipt is printed **to the user**.
                17. The ATM delivers the cash **to the user**.

Final states : The ATM is ready to serve. The user has cash. The user has ***the user's*** card. The user has a receipt.

Table 3 : Flow of actions after linguistic clarification and completion.

5.4. Action dependency completion

In addition to the linguistic completion illustrated above, completion is achieved by using action dependency rules. In the use case model, atomic actions are refined by several sub-types : service request, service provision, information provision, information request, internal action, etc. Based on this typology, we defined *action dependency patterns* which state dependencies between several types of atomic actions. The non respect of these dependencies is captured in the situations of the completion rules CO8 to CO12.

Rule CO8, for example, states that the provision of a service S to an agent A from an agent B should be preceded in the flow of actions by an action of request of the service request S from A to B.

CO8 : ∀ V1, S, A, B : (Atomic Action(Name : V1, From Agent : B, To Agent : A, Parameter : S) ∧ (Type(V1) = 'Service Provision')) ∧ ¬(∃ V2 : Atomic Action (Name : V2, From Agent : A, To Agent : B, Parameter : S) ∧ (Type(V2) = 'Service Request') ∧ Follow(V1, V2)) → ASK(« Complete service provision V1 with the anterior action :... (service request S from A to B) »)

Similarly, any service request should be followed by a service provision, and so on with information requests and provisions. These patterns are exploited in rules CO9, CO10, and CO11 which are similar to CO8.

Another action dependency pattern establishes the dependency between an alternative action and the action of verification of the corresponding flow condition. The rule CO12 corresponds to the completion which is triggered when this dependency pattern is not respected in the flow of actions.

CO12 : ∀ C: (Conditional (Is Based on Flow Condition : C, Composed of : _) ∧ ¬(∃ V1 : Atomic action (Name : V1, From Agent : _, To Agent : _, Parameter : C)) ∧ Follow(C, V1)) → ASK(« Complete conditional action based on flow condition C with the anterior action :... (condition verification action) »)

Standalone requests or provisions of services and information can be identified and completed if necessary with their counterpart by the rules CO8 to CO11. Similarly, conditional actions can be coupled with a verification action through rule CO12. A systematic application of these completion rules on our flow of actions of table 4 leads to the following specification, in which the new elements are emphasised in bold.

1. The user inserts the user's card in the ATM.
2. The ATM checks if the card is valid.
3. If the card is valid
    4. Then
    5. A prompt for code is given by the ATM to the user.
    6. The user inputs the code in the ATM.
    7. **The ATM checks if the code is valid.**
    8. If the code is valid
    9. Then
        10. A prompt for amount is displayed by the ATM to the user.
        11. The users enters an amount in the ATM.
        12. **The ATM checks if the amount is valid .**
        13. If the amount is valid
            14. Then
            15. The ATM ejects the card to the user.
            16. **The ATM asks the user if the user wants a receipt.**
            17. **The user enters a choice in the ATM.**
            18. If a receipt was asked by the user to the ATM
            19. Then
                20. The ATM prints a receipt to the user.
                21. The ATM delivers the cash to the user.

Final states : The ATM is ready to serve. The user has cash. The user has the user's card. The user has a receipt.

Table 4 : Flow of actions after the action dependencies completion.

Let us comment the incremental changes occurred between tables 3 and 4. As completion rules are based on types of actions, the use case author is first asked to provide atomic action typing. Mr. Bean thus identifies that :

- the *''insert card''* action is a *service request,*
- the *''check if the card is valid''* action is an *internal action of verification,*
- the *''give prompt for code''* and *''display prompt for amount''* actions are *information requests,*
- the *''input code''* and *''enter amount''* actions are *information provisions,*
- the *''eject card''*, *''print receipt''* and *''deliver cash''* actions are *service provisions.*

The *''print receipt''* service provision action, does not have a corresponding service request. Faced to this observation, Mr. Bean realises that the exchange of information between the user and the ATM is more complex than he initially stated. Thus he decides insert the following sentence in the flow of actions : *''The ATM asks the user if he wants a receipt and then the user enters a choice''.* Using the linguistic analysis and mapping rules, the sentence is then converted into the two atomic actions 16 and 17 of table 4. Indeed, the linguistic completion and clarification are also performed, as presented in previous section. This has led to replace *''he''* by *''the user''* in 16, and to complete 17 with a destination : *''in the ATM''.*

The application of the completion rules leads also to acknowledge that some parts of the specification are complete with respect to action dependency patterns. For example, the two *''display prompt''* actions are information requests from the ATM to the user which are followed by the provision of the requested information (code and amount). Thus, the description is already complete with respect to this exchange of information.

Note also that the use case author may decide not to apply the suggested completion. For example, the service provision *''ATM ejects the card to the user''* should be preceded by a request for card return. However Mr. Bean decides this service request is not sensible ; implicitly the user wants his card back at the end of the transaction since he enters it in the ATM.

There are now four flow conditions in the specification : (1) *''if the card is valid''*, (2) *''if the code is valid''*, (3) *''if the amount is valid''* and (4) *''if a receipt is asked by the user to the ATM''.* Two of them have a preceding action for checking the flow condition (1 and 4) but two do not (2 and 3). Asking Mr. Bean to verify the need for two new actions for the condition checking leads to complete the episode specification with *''(6) the ATM checks if the code is valid''* and *''(11) the ATM checks if the amount is valid''.*

5.5. Reasoning on flow conditions

A flow of actions issued from a scenario description naturally involves nested flow conditions. For instance, in the current version of the normal episode of the ATM use case, there are four nested flow conditions :

```
1- if the card is valid
    2- if the code is valid
        3- if the amount is valid
            4- if a receipt is asked by the user to the ATM
```

As a consequence of the scenario definition, this flow of actions together with the flow conditions constitutes a pathway that permits the user to reach successfully one of the possible final states of the episode. In the case above, the flow of actions leads to an happy ATM user with the cash, his card and a receipt. As presented in section 2, the normal episode may comprise several pathways, and be complemented by exceptional episodes. Each of them permits to reach one of the possible final states. We believe that reasoning on the nested flow conditions of the initial scenario description can help to discover new exceptional episodes, and new pathways of the normal case. The emergence rules based on flow conditions support this reasoning. These rules raise the questions of what happens if the flow conditions do not hold.

Based on the flow conditions of a given pathway the REM1 rule calculates the combinations of negative conditions that should be investigated. EM1 generates four cases in our example.

EM1 : $\forall N, \exists U, C$ *:*

NumberOfFC (ActionEpisode (EpisodeName : N, Use Case : U),n) $\wedge (\forall k \in \{0..\ n\text{-}1\}) \rightarrow$

ASK(«type of the Case $((\wedge_{j=1..k} C_j) \wedge \neg C_{k+1})$ *» ,type)* $\wedge$

GENERATE(CaseToConsider(_ , Use Case : U ,Type : type, FlowCondition :C ,Case : $((\wedge_{j=1..k} C_j) \wedge \neg C_{k+1})$ *, EpisodeName : N))*

The list of cases (1) to (4) is presented to Mr. Bean who is asked to type each of them (responses are in bold).

(1)	*card is not valid*	**: exceptional**
(2)	*(card is valid) & (code is not valid)*	**: normal**
(3)	*(card is valid) & (code is valid) & (amount is not valid)*	**: normal**
(4)	*(card is valid) & (code is valid) & (amount is valid) & (receipt is not asked)*	**: normal**

For each case, a new iteration in the specification process starts. This includes the activities 2, 3 and 4 presented in section 4 : the textual scenario description provided by the author, its analysis and completion and its integration in the use case specification. There are two kinds of integration according to the type of case : the integration of an exceptional case, and the integration of a new course of actions in the normal episode. The integration of an exceptional case is simple and consists in relating the exception to an action of the normal episode. The integration of a new normal course of actions in the normal episode requires to embed the new flow of actions in the current episode. Let us illustrate the process with the cases 1 and 4.

The *style* and *contents* guidelines proposed to the use case author during the capture of these new scenarios are the same as the ones used to capture the initial scenario. In addition, we offer a *copy & paste* facility which enables to duplicate in the scenario under writing, a course of actions which already exists in the specification. This facility is also used by our tool to provide the author with a partially written scenario description and ask him to complete it. The following text illustrates this step of the process for case (1). Mr. Bean's writing is in bold whereas the text automatically generated by the guidance tool is in italics, the use of the copy and paste functionality is mentioned as a comment between the signs ''/*'' and ''*/''.

*/*copy & paste action(s) 1 to 2 of the normal episode */*

The user inserts the user's card in the ATM.

The ATM checks if the card is valid.

If ⌉(the card is valid)

Then **the ATM displays an « invalid card » message and then the card is ejected.**

Final states **: The ATM is ready and the user has his card.**

The NL text is analyzed and completed in a similar way as the one illustrated for the initial scenario text. Then, the exceptional episode specification is generated as follows.

Exceptional case of the use case named Cash1

Name **: InvalidCard**

Occurrence condition **: When ⌉(the card is valid).**

Where **: the action 3 of the NormalCase episode.**

Action :

1. The ATM displays an « invalid card » message to the user.
2. The card is ejected by the ATM to the user.

Final states : The ATM is ready. The user has a card.

Proceeding in the same way with the case number 4, leads Mr. Bean describe the flow of actions when the receipt is not asked by the user to the ATM.

---

*/*copy & paste action(s) 1 to 17 of the normal episode */*

The user inserts the user's card in the ATM.

...

The ATM asks the user if the user wants a receipt.

The user enters a choice in the ATM.

If ⌉(a receipt is not asked by the user to the ATM)

then

/*copy & paste action 21 of the normal episode */

The ATM delivers the cash to the user.

Final states : **The ATM is ready and the user has cash and his card.**

Note that the *copy & paste* functionality is used in this case by Mr. Bean to introduce the action 21 of the normal episode in the current scenario.

The analysis and completion steps are performed to obtain a specification that can be integrated in the use case specification. The integration step stands in two parts : the integration of the final states and the integration of the actions. The integration of the final states leads to add the keyword « sometimes » when a final state exists in the normal episode but not in the new pathway and vice versa. This results in adding « sometimes » before the final state *''the user has a receipt''* in the normal episode. Then the integration rules IN3 to IN10 are applied to re-organize the course of actions of the normal episode. Let us illustrate the integration of case (4) with rule IN3.

IN3 : ∀ S, ∃ U, A,C,M, F : EpisodeStructure (Episode :S, Use Case : U, Action : A) ∧ CaseToConsider (Episode : S, Use Case : U , Type : 'Normal', FlowCondition :C, _ , EpisodeName : M) ∧ FinalStateS(Episode : S, Use Case U, Final State : F) ∧ IncludesAtEnd (AMC, AS) ∧ ⌉ Includes (AS, AMC) →
REPLACEALTERNATIVE (AM, FlowCondition : ⌉C, Action : Sequence (Alternative (FlowCondition : ⌉ C, Action : Diff (AMC,AS),-), Action : CommonBloc (AMC, AS)))]

Note : this rule makes use of three aliases AM for ActionEpisode (EpisodeName : M, Use Case : U), AMC for CopyThen(Action: AM, FlowCondition : ⌉C), and AS for CopyThen(Action : A, FlowCondition : C).
Moreover, three predicates are used. The Diff (A1,A2) function returns an action which is described in A1 but not in A2. The CommonBloc (A1, A2) function returns an action which is described in both A1 and A2. The REPLACEALTERNATIVE (A1, C1, Anew) predicate permits to replace in the flow of actions A1 the alternative action embedding the condition C1 by the action Anew.

The rule IN3 applies when the new actions to be integrated are placed after the ''then'' part of the considered alternative. In this case, the common actions of the two episodes are placed after the alternative which is let without ''else'' part. In our case, the rule IN3 is applied for the condition *"if the receipt is asked by the user to the ATM"*. A sequence composed of the alternative *''if the receipt is asked by the user to the ATM'',* and of the *''the ATM delivers cash to the user''* atomic action is generated. This means that the cash is delivered, independently of a request for receipt.

The integration of all the normal flows of actions, namely cases (2), (3) and (4) leads to the normal episode of table 5. An asterisk marks a flow condition related to an exceptional episode. Additional elements corresponding to the application of this step are in bold.

Normal episode

name : NormalCase

action :

```
1. The user inserts the user's card in the ATM.
2. The ATM checks if the card is valid.
3. If the card is valid*
    4. Then
    5. Repeat
        6. A prompt for code is given by the ATM to the user.
        7. The user inputs the code in the ATM.
        8. The ATM checks if the code is valid.
    9. Less than four times and until the code is valid
    10. If the code is valid before the fourth time
        11. Then
        12. Repeat
            13. A prompt for amount is displayed by the ATM to the user.
            14. The users enters an amount in the ATM.
            15. The ATM checks if the amount is valid.
        16. Less than four times and until the amount is valid
        17. If the amount is valid before the fourth time
            18. Then
            19. The ATM ejects the card to the user.
            20. The ATM asks the user if the user wants a receipt.
            21. The user enters if the user wants a receipt.
            22. If a receipt is asked by the user to the ATM
                23. Then
                24. the ATM prints a receipt to the user.
            25. The ATM delivers the cash to the user
```

Final states : The ATM is ready to serve. The user has cash. The user has a card. Sometimes, the user has a receipt.

Table 5 : Version 4 of the Normal episode.

The same reasoning can be recursively applied to the new flow conditions (10) and (17). This leads to the emergence and specification of two exceptional cases, namely 'InvalidCode' and 'InvalidAmount' that are described in the appendix. The appendix presents the specification of the ''withdraw cash'' use case as it stands at the end of the process developed in this section. Notice that, for space considerations, we have not applied all the available rules and therefore, the use case family is not complete.

6. Conclusion

Scenarios and use cases as 'design engines' become more and more popular. Part of their popularity seems due to their informality. In order to cope with this situation, the paper has presented an approach to support the use of natural language in the capture of textual scenarios and the construction of use case specifications. Scenarios are narrative descriptions of flows of actions whereas a use case specification is a complete description of all possible normal and non normal flows of actions. The linguistic foundation of our approach is a Case Grammar which is tailored to our use case model. The grammar offers both semantic case patterns and linguistic structures connected to the use case concepts. The latter provides numerous surface structures for expressing in natural language, atomic actions, states, flow conditions etc. i.e. the elements which compose the use case specification. The former expresses the semantics of the use case contents. The case grammar permits us to go beyond the simple solution of sentence templates by providing tools to analyze textual scenarios and support their clarification and transformation into a non ambiguous textual use case specification.

Besides the linguistic support, the approach provides construction guidance. Construction guidance is based on use case model knowledge and takes the form of rules which encapsulate knowledge about types of action dependency, relationships between actions and flow conditions, properties of objects and agents, etc. Based on this knowledge rules help discovering incomplete expressions, missing elements, exceptional cases and episodes in the use case specification. They support the progressive integration of scenarios into a complete use case specification.

The guided process was illustrated with the ATM case study. However, we focused on the linguistic support and restricted to the ATM-user interaction. Our current work consists in extending the guidance rules to support the emergence of system internal requirements on one hand, and system contextual requirements on the other hand. The former relates to internal system objects and behaviour whereas the latter deals with the interactions of the system with other systems and organisational agents. Examples of such requirements shall *be 'the ATM must keep track of all transactions'*, '*the ATM must be able to access to the user account balance'*, *'the ATM must have the weekly amount allowed per card type'*, '*the ATM must be able to identify a user who is a client of the bank'*, '*the ATM must have a clock'*. Meanwhile we are extending the current PROLOG implementation to support the entire set of rules.

7. References

[1] B. Belkhouche, J. Kozma, Semantic Case Analysis of Informal Requirements, *Proc. Of the 4th Workshop on the Next Generation of CASE Tools*, Enshede, The Netherlands, (1993) 163-181.

[2] B. Boguraev and K. Spark-Jones, A Note on a Study of Cases. In *Computational Linguistics*, Vol 13, n° 1-2 (1987) 65-68.

[3] 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).

[4] P. P-S Chen, English Sentence Structure and Entity Relationship Diagrams, in *Information Sciences* 29, (1987) 127-149.

[5] N. Chomsky, Deep Structure, Surface Structure and Semantic Interpretation, (Steinberg & Jakobovits 1971), 183-216.

[6] A. 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).

[7] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes and P. Jeremaes, *Object-Oriented Development: The FUSION Method* (Prentice Hall, 1994).

[8] B. Dano, H. Briand, F. Barbier, A Use Case Driven Requirements Engineering Process, In *Third IEEE International Symposium On Requirements Engineering (RE'97)*, Antapolis, Maryland (IEEE Computer Society Press, 1997).

[9] S.C. Dik, The Theory of Functional Grammar, Part I : The Structure of the Clause, Functional Grammar Series, Fories Publications, (1989)

[10] Dowty, Walls, Peters, Introduction to Montague Semantics, Reidel-Klawer, 1981.

[11] E. Dubois, P. Heymans, M. Jarke, K. Phol, K. Weidenhaupt, A. Sutcliffe and N.A.M. Maiden, Integration of Scenario Representations: Formalisation and Examples, ESPRIT Reactive Long Term Research Project 21.903 CREWS, Deliverable W4: Knowledge Representation Working Group (1997).

[12] T. Erickson, Notes on Design Practice: Stories and Prototypes as Catalysts for Communication, in John M. Carroll (ed.*), Scenario-Based Design: Envisioning Work and Technology in System Development* (John Wiley and Sons, 1995) 37-59.

[13] C. Fillmore, The Case For Case, in Holt, Rinehart and Winston (eds.*), Universals in Linguistic Theory* (Bach & Harms, 1968) 1-90.

[14] J.D. Gould, How to design usable systems, in M. Helander (ed.), *Handbook of Human-Computer Interaction* (Elsevier Science, 1988) 757-790.

[15] I. Jacobson, M. Christerson, P. Jonsson and G. Oevergaard, *Object Oriented Software Engineering: a Use Case Driven Approach* (Addison-Wesley, 1992).

[16] I. Jacobson, The Use Case Construct in Object-Oriented Software Engineering, in John M. Carroll (ed.), *Scenario-Based Design: Envisioning Work and Technology in System Development* (John Wiley and Sons, 1995) 309-336.

[17] I. Jacobson, M. Ericsson and A. Jacobson, *The Object Advantage, Business Process Reengineering with Object Technology* (Addison-Wesley Publishing Company, 1995).

[18] M. Jarke, K. Pohl, P. Haumer, K. Weidenhaupt, E. Dubois, P. Heymans, C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyte, A. Sutcliffe, N.A.M. Maiden and S. Monicha, Scenario Use in European Software Organisations - Results from Site Visits and Questionnaires, ESPRIT Reactive Long Term Research Project 21.903 CREWS, Deliverable W1: Industrial problem capture Working Group (1997).

[19] P. Johnson, H. Johnson and S. Wilson, Rapid Prototyping of User Interfaces driven by Task Models, in John M. Carroll (ed.*), Scenario-Based Design: Envisioning Work and Technology in System Development* (John Wiley and Sons, 1995) 209-246.

[20] J. Karat, Scenario Use in the Design of a Speech Recognition System, in John M. Carroll (ed.*), Scenario-Based Design: Envisioning Work and Technology in System Development* (John Wiley and Sons, 1995) 109-135.

[21] K. Koskimies and H. Mossenbock, Scene: Using Scenario Diagrams and Active Text for illustrating Object-Oriented Programs, in *Proc. of ICSE-18* (1995) 366-375.

[22] K. Koskimies, T. Mannisto, T. Systa, and J. Tuomi, On the Role of Scenarios in Object-Oriented Software Design, Technical report, Series of Publications A (A-1996-1), Department of Computer Science, Univerity of Tampere, Finland (1996).

[23] M. Kyng, Creating Contexts for Design, in John M. Carroll (ed*.), Scenario-Based Design: Envisioning Work and Technology in System Development* (John Wiley and Sons, 1995) 85-107.

[24] J.C.S. do Prado Leite, G. Rossi, F. Balaguer, A. Maiorana, G. Kaplan, G. Hadad and A. Oliveros, Enhancing a requirements baseline with scenarios, In *Third IEEE International Symposium On Requirements Engineering (RE'97),* Antapolis, Maryland (IEEE Computer Society Press, 1997) 44-53.

[25] 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.

[26] B. A. Nardi, The Use of Scenarios in Design, in *SIGCHI Bulletin* 24(4).

[27] J. Nielsen, Scenarios in Discount Usability Engineering, in John M. Carroll (ed*.), Scenario-Based Design: Envisioning Work and Technology in System Development* (John Wiley and Sons, 1995) 59-85.

[28] C. Potts, K. Takahashi and A.I. Anton, Inquiry-based Requirements Analysis, *in IEEE Software* 11(2) (1994) 21-32.

[29] D.A. Rawsthorne, Capturing functional Requirements through Object Interactions, in *Proceedings of ICRE '96* (IEEE, 1996) 60-67.

[30] B. Regnell, K. Kimbler and A. Wesslen, Improving the Use Case Driven Approach to Requirements Engineering, in *the Second IEEE International Symposium On Requirements Engineering*, York, England (I.C.S. Press, March 1995) 40-47.

[31] S.P. Robertson, Generating Object-Oriented Design Representations via Scenarios Queries, in John M. Carroll (ed*.), Scenario-Based Design: Envisioning Work and Technology in System Development* (John Wiley and Sons, 1995) 279-308.

[32] C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyte, A. Sutcliffe, N.A.M. Maiden, M. Jarke, P. Haumer, K. Pohl, E. Dubois and P. Heymans, A Proposal for a Scenario Classification Framework, ESPRIT Reactive Long Term Research Project 21.903 CREWS, Deliverable I1: Initial Integration Workpackage (1997).

[33] C. Rolland, C. Proix, A Natural Language Approach for Requirements Engineering, in *Advanced Information Systems Engineering* (P. Loucopoulos ed.), Springer Verlag, (1992), 257-277.

[34] M.B. Rosson and J.M. Carroll, Narrowing the Specification-Implementation Gap, in John M. Carroll (ed*.), Scenario-Based Design: Envisioning Work and Technology in System Development* (John Wiley and Sons, 1995), 247-278.

[35] K.S. Rubin and A. Goldberg, Object Behaviour Analysis, *Communications of the ACM* 35(9) (1992) 48-62.

[36] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, (Prentice Hall, 1991).

[37] J. Rumbaugh., Getting started, *Journal of Object-Oriented Programming* (7) (1994) 8-23.

[38] J. Rumbaugh and G. Booch, Unified Method, Notation Summary Version 0.8 (Rational Software Corporation, 1996).

[39] B. Scalzo, UPROAR - User processes reveal Objects and Requirements, in *Proc. of the OOPSLA'95 Workshop on Use Cases*, (1995).

[40] R. Schanck, Identification of Conceptualisations Underlying Natural Language, *in Computer Models of Thought and Language*, (Shanck & Colby, Freeman, San Francisco, 1973) 187-247.

[41] R. Simmons, Semantic Networks : their Computation and Use for Understanding English Sentences, in *Computer Models of Thought and Language,* (Schanck & Colby, Freeman, San Francisco, 1973) 63-113.

[42] S. Somé, R. Dssouli, J. Vaucher, Towards an Automation of Requirements Engineering Using Scenarios, *Journal of Computing and Information 2 (1), special issue : ICCI'96, 8th Int. Conf. Of Computing and Information, Waterloo, Canada* (1996), 1110-1132.

[43] John F. Sowa, *Conceptual Structures, Information Processing in Mind and Machine* (Addison-Wesley, The Systems Programming Series, 1984).

[44] K. Weidenhaupt, K. Pohl, M. Jarke, P. Haumer, Scenario Usage in System Development : a Report on Current Practice, ESPRIT Reactive Long Term Research Project 21.903 CREWS, Deliverable W1-D2, (1997).

[45] J. Whiteside, J. Bennett and K. Holtzblatt, Usability Engineering : Our experience and evolution, in M. Helander (ed.), *Handbook of Human-Computer Interaction* (Elsevier Science, Amsterdam, 1988) 791-818.

[46] Y. Wilks, Good and Bad Arguments about Semantic Primitives. Report n° 42, Department of Artificial Intelligence, University of Edinburgh (1977).

[47] R. Wirfs-Brock, Designing Objects and their Interactions: A Brief Look at Responsability-driven Design, in John M. Carroll (ed.*), Scenario-Based Design: Envisioning Work and Technology in System Development* (John Wiley and Sons, 1995) 337-360.

8. Appendix

The input text :

The user inserts his card in the ATM. The ATM checks if the card is valid. Then, a prompt for code is given and the user inputs the code. If the code is valid, a prompt for amount is displayed. The user enters an amount. If the amount is valid, the ATM delivers the cash, but before the ATM ejects the card and a receipt is printed if this one was asked.

The final states are : The user has the cash his card, and a receipt. The ATM is ready to serve.

The contextual information of the use case obtained after the process described in section 5 is the following :

Use case Cash1 **belongs** to the bank application

Its initiating agent is the user who has the **goal** of withdraw cash.

Assuming that an automated teller machine is used to provide cash to a user having a credit card.

It requires the user to have a credit card and the automated teller machine to be ready.

The secondary agent participating to the use case is the automated teller machine called ATM

The resources used in this use case are : a card, code, amount, receipt, cash.

The contents of the use case is composed of one normal episode and three exceptional episodes.

Normal episode of the use case *Cash1*

name : NormalCase

action :

1. The user inserts the user's card in the ATM.
2. The ATM checks if the card is valid.
3. If the card is valid
    4. Then
    5. Repeat
        6. A prompt for code is given by the ATM to the user.
        7. The user inputs the code in the ATM.
        8. The ATM checks if the code is valid.
    9. Less than four times and until the code is valid.
    10. If the code is valid before the fourth time
        11. Then
        12. Repeat
            13. A prompt for amount is displayed by the ATM to the user.
            14. The users enters an amount in the ATM.
            15. The ATM checks if the amount is valid.
        16. Less than four times and until the amount is valid
        17. If the amount is valid before the fourth time
            18. Then

19. The ATM ejects the card to the user.
20. The ATM asks the user if the user wants a receipt.
21. The user enters if the user wants a receipt.
22. If a receipt is asked by the user to the ATM
    23. Then
    24. The ATM prints a receipt to the user.
25. The ATM delivers the cash to the user

Final states : The ATM is ready to serve. The user has cash. The user has a card. Sometimes, the user has a receipt.

Exceptional case of the use case *Cash1*

name : InvalidCard

occurrence condition : When ⌉(the card is valid).

Where : the action 3 of the NormalCase episode.

action :

1. The ATM displays an « invalid card » message to the user.
2. The card is ejected by the ATM to the user.

Final states :

The ATM is ready. The user has a card.

Exceptional case of the use case *Cash1*

name : InvalidCode

occurrence condition : When ⌉(the code is valid before the fourth time).

Where : the action 9 of the NormalCase episode.

action :

1. The ATM displays an « invalid code » message to the user.
2. The ATM keeps the card.

Final states :

The ATM is ready. The ATM has the user’s card.

Exceptional case of the use case *Cash1*

name : InvalidAmount

occurrence condition : When ⌉(the amount is valid before the fourth time).

Where : the action 15 of the NormalCase episode.

action :

1. The ATM displays an « invalid amount » message to the user.
2. The ATM keeps the card.

Final states :

The ATM is ready. The ATM has the user’s card.

Textual form of the use case :

Use case Cash1 **belongs** to the bank application

Its initiating agent is the user who has the **goal** of withdraw cash.

Assuming that an automated teller machine is used to provide cash to a user having a credit card.

It requires the user to have a credit card and the automated teller machine to be ready.

The secondary agent participating to the use case is the automated teller machine called ATM

The resources used in this use case are : a card, code, amount, receipt, cash.

The **normal case** of the use case *Cash1* **named** NormalCase **is described by** :

The user inserts the user's card in the ATM. The ATM checks if the card is the card is valid. If the card is valid repeat less than four times and until the code is valid ; a prompt for code is given by the ATM to the user, the user inputs the code in the ATM, and the ATM checks if the code is valid.

If the code is valid before the fourth time, repeat less than four times and until the amount is valid ; a prompt for amount is displayed by the ATM to the user, the user enters an amount in the ATM, the ATM checks if the amount is valid.

If the amount is valid before the fourth time then the ATM ejects the card to the user. The ATM asks the user if the user wants a receipt. The user enters if the user wants a receipt in the ATM.

If a receipt is asked by the user to the ATM then the ATM prints a receipt to the user.

Even if the receipt is not asked by the user, the ATM delivers the cash to the user.

This episode permits **to reach the following states** : the user has the card, the user has cash, sometimes the user has a receipt, the ATM is ready to serve.

The exceptional case of the use case Cash1 named InvalidCard **occurs when** the card is not valid**. The referenced action in the** NormalCase **episode is «** If the card is valid**»**

It is described by : the ATM displays an « invalid card » message to the user. The card is ejected by the ATM to the user.

This episode permits **to reach the following states** : the ATM is ready, the user has a card.

The exceptional case of the use case Cash1 named InvalidCode **occurs when** the code is not valid before the fourth time**. The referenced action in the** NormalCase **episode is «** If the code is valid before the fourth time **»**

It is described by : the ATM displays an « invalid code » message to the user. The ATM keeps the card.

This episode permits **to reach the following states** : the ATM is ready, the ATM has the user's card.

The exceptional case of the use case Cash1 named InvalidAmount **occurs when** the amount is not valid before the fourth time**. The referenced action in the** NormalCase **episode is «** If the amount is valid before the fourth time **»**

It is described by : the ATM displays an « invalid amount » message to the user. The ATM keeps the card.

This episode permits **to reach the following states** : the ATM is ready, the ATM has the user's card.

Glossary of the use case :

It comprises the agents, resources and actions used in the use case *cash1***.**

agents :

- a user
- an automated teller machine called ATM.

resources :

- a card
- a prompt for code
- a code
- a prompt for amount
- an amount
- a receipt
- cash
- an « invalid card » message
- an « invalid code » message
- an « invalid amount » message

actions :

- **internal actions :**
  - check
  - keep
- **communication actions :**
  - insert
  - give
  - input
  - display
  - enter
  - eject
  - ask
  - print
  - deliver