Requirements Eng (2000) 5:38–61

Requirements Engineering

A Scenario Construction Process

Julio Cesar Sampaio do Prado Leite[a], Graciela D. S. Hadad[b], Jorge Horacio Doorn[c] and Gladys N. Kaplan[d]

[a]Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil; [b]Universidad Nacional de La Plata, Buenos Aires, Argentina; [c]Universidad Nacional del Centro de la Provincia de Buenos Aires, Buenos Aires, Argentina; [d]Universidad de Belgrano, Buenos Aires, Argentina

Scenario is a description technique that has been attracting a lot of attention from practitioners and from researchers. Several disciplines have been using scenarios for some time now, but recently the information system community has dedicated special attention to the possibilities that this description technique provides to enhance understandability of task-related descriptions and communicability among stakeholders. This paper aims its attention at a particular scenario construction process, but while doing so it tackles important problems regarding scenario management, in particular scenario organisation. Our scenarios are described in a structured way, using a simple conceptual model together with a form-oriented language. We have been using this representation for some time now, and our results are based on several case studies conducted with real-world problems.

Keywords: Organisational context; Requirements elicitation; Scenario management; Scenario modelling; Stakeholders

1. Introduction

The necessity of ensuring good understanding between requirements engineers and clients/users has motivated the research and development of methods that allow better results in the collaboration among all participants in the requirements definition process. Requirements engineers should understand, model and analyse the application domain where the software will run and the clients/users must validate whether or not the engineers' vision is correct. Our research uses the idea of scenario as a medium to achieve this goal.

The word scenario, defined as 'the plot of a motion picture' in the Merrian–Webster Dictionary, has been used for a long time in several areas of different disciplines (military strategy, decision making, marketing, economy). The information system area started using scenarios in the human–computer interface and recently in software engineering. Jarke, Bui and Carroll [50] provide an overview of scenario research in a broader view, both from an information system point of view as well as from a management (decision theory) view. They point out that the effectiveness of the use of scenarios in several disciplines is fundamentally due to their capability of stimulating thinking. Scenario provides a situated task vision together with an effective way of communication among the actors involved in the subject of study.

For us, scenarios describe situations that happen in the Universe of Discourse (UofD).[1] Carroll [5], taking into account usage aspects, believes that scenarios allow us to know the problem, to unify criteria, to gain clients/users compromise, to organise the involved details and to train new participants. The use of scenarios, as a way of understanding the domain in which the software will perform, has been recommended by several authors [6–9] and those proposals became very important for extending the use of scenarios in real practice. However, a detailed analysis of the recommendations from the

Correspondence and offprint requests to: Julio Cesar Sampaio do Prado Leite, Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio, Rua Marquês de São Vicente 255, 22451-041 Gávea-RJ, Rio de Janeiro, Brazil. Email: julio@inf.puc-rio.br

[1]'The overall context in which software will be developed and operated. The UofD includes all the sources of information and all the people related to the software. It is the reality trimmed by the set of objectives established by those demanding a software solution.' [1, 2]. We use the term *Universe of Discourse* with the same meaning given by Michael Jackson [3] to *Application Domain*. Loucopoulos [4] also uses both terms as synonyms.

literature shows some degree of dispersion and contradiction in the use of scenarios. Recent work of the CREWS project confirms this observation [10,11].

This lack of precision of when and how scenarios should be used has spread to the engineers who are using these techniques in the field. Thus, most developers see scenario creation more as a craft than as an engineering task. Recent studies concerning the use of scenarios in industrial projects [11] have clearly proved this fact and pointed out the necessity of more detailed definitions on scenario construction to increase their use in real situations. This is in complete agreement with Rolland et al. [12] and Sutcliffe [13], who think that there is little understanding about the usage and production of scenarios.

The variety of interpretations, syntax and construction mechanisms for scenarios goes as far as showing up basic contradictions. For instance, with respect to the scenario construction process there is no agreement whether scenarios should be built in a top-down or in a bottom-up fashion. We believe that part of the problem of such diversity in the scenario literature is due to the fact that the scenarios building process should be neither top-down nor bottom-up.

The present work proposes a strategy for the creation and use of scenarios[2] as a grounding representation for the requirements engineering process. The strategy is built on the assumption that scenarios must rely on natural language as a means of communication among stakeholders, in particular among clients/users and requirements engineers. The strategy also relies on the idea of scenario evolution [14], starting from what Rolland et al. classified as *organisational context* [10], which aims at the 'broad picture of how the work gets done' [10, p. 30], and what Jarke et al. [50] called environmental scenarios. Our scenarios are neither specifications nor requirements, they are auxiliary descriptions for the process of requirements definition. They provide a knowledge source where requirements may be found and specifications may be based upon.

This paper details a scenario construction process. This process addresses two important problems related to scenario management [50]: organisation and quality. The organisation problem arises as we start to deal with a large number of scenarios. The quality problem refers to the reliability of the descriptions presented as scenarios. Regarding the first problem, we provide an innovative middle-out strategy that systematises the construction process using typed relationships and operational heuristics. Regarding quality, we provide policies and procedures for detecting defects and errors in scenarios. It is important to stress that our proposal is based on a cumulative experience on the use of scenarios. For the last four years, we have analysed more than 20 software projects that used scenarios in the requirements process. In total we have analysed more than 400 different scenarios.

We understand that describing the process, giving data on its use and comparing it with other work contributes to increase the knowledge about scenario management. We have organised the paper into six sections. Section 2 provides a survey of existing scenario construction processes. In section 3 we present our strategy, giving emphasis to its reliance on natural language. Detail on how scenarios are put together is given in section 4. Section 5 reports on our experience in producing scenarios. In conclusion we state that our results are important to scenario management by comparing our work with previous work, as well as pointing to future research.

2. Different Approaches for Scenario Construction

There are several styles[3] in which scenarios are built. such as textual narrative, storyboards, video mock-ups and written prototypes [5,7,8,9,15]. This section stresses the distinction between top-down and bottom-up strategies by quoting some of the construction heuristics proposed by the literature. We believe that this review is important since our proposal, a middle-out strategy, is perceived by us to be a major contribution. This belief is based on our own experience of previously using top-down and bottom-up strategies. In the following paragraphs we present different quotations from the literature.

Even if it is not clearly stated, Booch [16] seems to adhere to a top-down approach since: 'The most complex application can be characterized in terms of a few dozen primary scenarios. Primary scenarios model the central problem. A Primary scenario represents some fundamental system function. Secondary Scenarios represent some variation on the theme of a primary scenario. The entire desired behavior of a software system can be captured through a web of scenarios, much the same way as a storyboard does in making a movie.'

Booch's primary scenarios are seen as relevant scenarios by Firesmith [17]; they are linked together by a scenario lifecycle diagram: 'A scenario lifecycle diagram is used to document the valid interactions

[2] 'The process dimension of scenarios is seldom considered in the literature' [10, p. 36].

[3] The form view in the CREWS framework [10]. In this paper we consider 'use cases' as one of the several styles of scenarios.

among the top level scenarios of a system or assembly.' And 'Because scenarios are functional abstractions, there is often a strong tendency to functionally decompose them.'

Sutcliffe's [18] proposal may also be seen as a top-down approach taking into account the heuristics: '1. Initial requirements capture and domain familiarization. A grounding scenario is developed based on the preliminary domain analysis, 2. Specification and development of the concept demonstrator (early prototype). Design Rationale diagrams are used to explain design options at the key points, 3. Requirements Analysis – validation session to critique the concept demonstrator.'

The scenario formalisation process using regular grammars in a tree of scenarios described by Hsia et al. [19] is also an example of a top-down approach.

On the other hand, Dano et al. [20] are closer to a bottom-up approach in their proposal 'to collect and describe use cases based on a tabular notation' and then 'to create Petri Nets for each use case in order to bring the necessary formalism to the analyst' and finally 'to set up formal inter use cases links to obtain a global description of the application.'

Robertson [21] identifies the need for linking scenarios: 'Since a single scenario gives partial information, scenario-based analysis involves the observation of a set of scenarios.' And 'The systematic question-asking technique helps to create a bridge between the specific scenario events and the general situations that enable, cause, or explain the events.'

Potts et al.'s *Inquiry Cycle Model* [15] may also be included in the bottom-up approach: 'Scenarios are represented at two levels of detail: 1. Episodes or phases that are sequences of fine-grained actions, 2. Families of Scenarios using Jacobson's uses relationship.'

It is also useful to watch the construction of use cases, where top-down approaches may be found, such as in Wirfs-Brock's heuristics [22]: '1. Determine the software system (boundaries, scope, identify actors, identify system interfaces, develop first-cut subsystem partitioning), 2. Stereotype the actors (active or passive, roles), 3. Determine system use-cases (decompose system into discrete chunks meaningful to both businesspeople and developers), 4. Construct conversations (sequence of interactions between actors and the system for each use-case).'

Also, Oberg et al.'s [23] proposal corresponds to a top-down approach, since they wrote: '1. Problem analysis', '2. Understanding stakeholder needs', that involves 'Find actors and use cases' and 'Outline the use-case model', '3. Defining the system', that involves the tasks: 'Refine the use-case model' and 'Describe use cases', '4. Managing the scope of the project', that involves: 'Prioritize use cases' and 'Model a use-case view of the system architecture', '5. Refining the system definition' that involves: 'Detail use cases', and finally '6. Managing changing requirements.'

Schneider and Winters' [24] proposal consists of four main primary phases following a top-down approach: '1. Inception phase: identify actors and high-level use cases are developed to help scope out the project; 2. Elaboration phase: more detailed use cases are developed, they are used to create the plan for next phase. Product of this phase: detailed primary scenarios and secondary scenarios; 3. Construction phase: use cases are used as a starting point for design and for developing test plans; 4. Transition phase: use cases can be used to develop user guides and training.'

An intermediate approach closer to top-down is that one of Constantine [25], where the process for generating uses cases involves the following: 'Identify candidate use cases based on user role model and brainstorming: generate a list of use cases; Sketch use case map based on initial understanding (relationships among use cases); Develop use case narratives in structured form; Reduce concrete use cases to essential form; Switch between narratives and map when needed.'

On the other hand, Jacobson [26,27] follows a strict bottom-up approach: 'We first describe the basic use cases and then describe how to enhance them to allow more advanced use cases, some of which may depend on the basic ones. Abstract use cases should evolve from concrete use cases, not the other way round. Extends association let us capture the functional requirements of a complex system, in the same way we learn about any new subject: First we understand the basic functions, then we introduce complexity.'

Gough et al. [28] follow an approach closer to the one proposed in this article regarding their heuristics: '1. Creation of natural language documents: project scope documents, customer needs documents, service needs documents, competitor analysis and market analysis. Get functional specifications of an existing system, 2. To Identify the main actors in the system, 3. To derive the Scenario headings, 4. To complete Scenario Description.'

In their conclusions Gough et al. [28] clearly establish: 'The need to provide a high level view of scenarios interaction as a means of addressing the complexity of a large number of scenarios.'

Weidenhaupt et al. [11] have found four types of scenario processes in practice: '(i) Top-down decomposition', '(ii) From black-box to white-box scenarios', '(iii) From informal to formal scenario definition' and '(iv) Incremental scenario development.'

There are several lines written about top-down or bottom-up, but we would like to point out Jackson's

view [3] on the matter: 'The first (difficulty) is that Top-Down enforces the riskiest possible ordering of decisions. The largest decision is the subdivision of the whole problem: it is the largest in scale and the largest in its consequences. Yet this decision is taken first, when nothing is yet known and everything remain to be discovered.' 'The second difficulty is that the real world hardly ever has a hierarchical structure. Instead there are many parallel and overlapping structures.'

With these excerpts from the literature we have shown that there is no agreement regarding how to build scenarios. Of course, if the functionality of the system is well known, the top-down approach could be used, while if there is not enough knowledge we need to extract it from the bottom up. In our process, where scenarios are first deployed to understand the UofD, we start from a bottom-up manner and later organise the scenarios using the middle-out strategy.

We would like to point out that our contribution should be viewed as a scenario engineering approach. Although we do not claim to have the final word on scenario construction, we firmly believe that the detailed description of our process contributes to the field and establishes grounds for further improvements.

3. A Natural Language-Based Approach

In this section we will focus on representation schemas that deal with scenarios. First, we give a general overview of our approach, which uses a controlled vocabulary based on the lexicon of the application. The following sections, 3.2 and 3.3, detail the representations we use for scenarios and for the lexicon.

3.1. Overview

Although we know that the requirements engineers should elicit, model and analyse requirements, we also know that these tasks are very much tangled, forming a complex web of relationships. Our work [2] does tackle these three tasks, but here we will be more concerned with the modelling aspect, in particular with an emphasis on the organisation aspect of modelling.

A scenario is a partial description of the application behaviour that occurs at a given moment in a specific geographical context – a situation [9]. Several authors have studied this technique [5,7–9,15]. Although each scenario describes a particular situation, none of them is entirely independent of the rest of the scenarios. Each scenario keeps a semantic relationship with other scenarios [16].

Scenarios accomplish different goals depending on the phase where they are used during the software development process. In the RE process, the scenario goals are:

- to capture the requirements;
- to provide a means of communication among stakeholders;
- to provide an anchor for traceability.

Scenarios are not created from the minds of the requirements engineers; their definition should be anchored on real situations. Our work anchors scenarios on the organisational context [10]. But how do we elicit such situations and model them accordingly? How do we avoid using a modelling scheme that does not fit the elicitation process? As such, we have the challenge of having to choose a modelling strategy that fits the process of eliciting and representing situations.

Using natural language for describing situations helps the validation with the client/user and complies with the goal of improving stakeholders' communication. The use of the Language Extended Lexicon (LEL) and scenarios for requirements elicitation and their utilisation throughout the entire software development process [2] allows for validation with the client/user. The main purpose of the lexicon is to capture the application vocabulary and its semantics, postponing the comprehension of the application functionality [29]. Scenarios are used to understand the application and its functionality: each scenario describes a specific situation of the application, centring the attention on its behaviour.

The rationale of the proposed approach was presented in Leite et al. [2] and the details of the first case study, based on the *Argentine Passports Issue System*,[4] were reported in Hadad et al. [30]. On the basis of these results, we conducted several other case studies, including the one used as an example in this article, which is a variation of the widely known *Meeting Scheduler* problem [31, 32].[5] This case was studied only for the RE process and consisted in modelling the meeting scheduler for the authorities of the Universidad de Belgrano, Buenos Aires.

Our scenario model has a decomposition organisation, but this does not imply that we use a top-down approach to build the scenarios. We do not use a bottom-up approach either. A truly bottom-up process would begin by discovering episodes, then building

[4]The past strategy consisted on building the LEL from the UofD. Afterwards, scenarios were constructed based on the UofD and using the vocabulary defined in the LEL. The lexicon and scenarios were then validated against the UofD and the scenarios were used to verify the lexicon.

[5]The problem proposed by van Lamsweerde is used for validating methods/techniques in RE.

sub-scenarios and finally integrating them in scenarios. A top-down process would begin building either one or few scenarios of the whole system, refining them later to obtain a succession of scenario sets with increasing levels of detail.

We based the construction of scenarios on the vocabulary of the Universe of Discourse (UofD). This vocabulary reflects the peculiar and most used words or phrases in the UofD and should be present in the LEL. The LEL is a representation of the symbols in the application language, and it is anchored on a very simple idea: understand the language of the problem, without worrying about understanding the problem.

Several authors [23,25] suggest the use of a glossary containing the vocabulary of the application. Our proposal is to build not just a glossary but a lexicon that involves the denotation and the connotation of every symbol discovered as a word or phrase relevant to the application domain. The purpose of constructing this lexicon is not only to enable a good communication and agreement between the clients/users and the engineering team but also to bootstrap the scenario construction process and to help their description facilitating the validation process. The use of the symbols of the lexicon in the scenarios makes it possible for these symbols to be a natural hyperlink between these two representation structures, a fundamental characteristic of our requirements baseline concept [2].

In real practice, of 15 projects analysed by the CREWS project [11] just one project, and to a lesser extent three others, had employed glossaries. This one project presented the practice of linking terms in the glossary to scenarios by means of a hypertext navigation. In our understanding this is an indication of the necessity that requirements engineers have to anchor their understanding on the application vocabulary.

A fundamental characteristic of our work [2] is that we anchor the natural language representation of scenarios on a previous lexicon of the application language. This characteristic is original and tackles the important problem of reducing ambiguity in natural language-based descriptions. As the scenario uses the LEL symbols, they became a hypertext, the lexicon symbols being the hyperlinks between these two representations.

In the following sections we detail the representation schemas used to describe the lexicon (LEL) and the UofD situations (scenarios). Again we stress that both representations are not intended to replace the requirements specification – their goal is to help the production of specifications.

3.2. Language Extended Lexicon (LEL)

The LEL [33] is a meta-model designed to help the elicitation and representation of the language used in the application. This model is centred on the idea that a closed description of language terms improves the comprehension of the UofD. It is a natural language representation that aims to capture the vocabulary of an application.

The lexicon main goal is to register signs (words or phrases) which are peculiar to the application domain. Each entry in the lexicon is identified by a name or names (case of synonyms) and has two descriptions (as opposed to the usual dictionary, which provides only one). The first, called Notion, is the usual one and its goal is to describe the denotation (defines 'what the symbol is') of the word or phrase. The second, called Behavioural Response, is intended to describe the connotation (describes 'how the symbols acts in the system') of the word or phrase, that is, it provides extra information about the context at hand. Entries are classified into four types according to its general use in the Universe of Discourse. The types are: Subject, Object, Verb and State.

Figure 1 presents the Language Extended Lexicon Model, while Fig. 2 shows an example of a symbol based on the *Meeting Scheduler* case.

While describing the symbols, two principles [29] have to be followed: the *principle of circularity*[6] that intends to maximise the use of symbols in the description of other symbols, and the *principle of*

LEL: representation of the symbols in the application domain language.
Syntax:
$\{Symbol\}_1^N$

Symbol: entry of the lexicon that has a special meaning in the application domain.
Syntax:
$\{Name\}_1^N + \{Notion\}_1^N + \{Behavioral\ Response\}_1^N$

Name: identification of the symbol. More than one represents synonyms.
Syntax:
Word | Phrase

Notion: denotation of the symbol. It must be expressed using references to other symbols and using a minimal vocabulary.
Syntax:
Sentence

Behavioral Response: connotation of the symbol. It must be expressed using references to other symbols and using a minimal vocabulary.
Syntax:
Sentence

where Sentence is composed only by Symbols and Non-Symbols, the latter belongs to the minimal vocabulary.

+ means composition, {x} means zero or more occurrences of x,
() is used for grouping, | stands for or and [x] denotes that x is optional

Fig. 1. The Language Extended Lexicon Model.

[6]This rule is also called 'principle of closure'.

Requester
Notion:
- person who invites attendees to a meeting.
- may be a participant.

Behavioral Response:
- defines the objective of the meeting, the subjects to be discussed, the attendees, the material to exhibit and the material to distribute.
- records the objective and the attendees in the agenda.
- performs the scheduling of meetings.
- organizes the meeting.
- records the substitute in the agenda.
- decides the cancellation of the meeting.
- decides changes in the meeting requirements.
- decides the moving of date.

Fig. 2. An example of an LEL symbol.

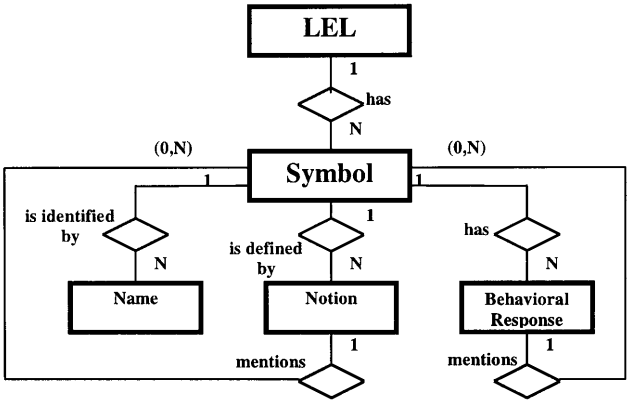


Fig. 3. Entity-relationship diagram for the Language Extended Lexicon Model.

minimal vocabulary that intends to minimise the use of symbols that are external to the lexicon. These external symbols must belong to a small subset of a natural language dictionary.[7] These rules stress the description of the vocabulary as a self-contained and highly connected hypertext [33].

In Fig. 3, we provide an entity-relationship [34] model of the LEL.

3.3. Scenarios

Our scenario model is a structure composed of the entities: *title*, *goal*, *context*, *resources*, *actors*,[8] *episodes* and *exceptions* and the attribute constraint (see Fig. 4). Actors and resources are an enumeration. Title, goal, context and exceptions are declarative sentences while episodes are a set of sentences expressed in a simple language that give an operational description of behaviour.

[7]For instance, in English the words in Collins Dictionary's COBUILD Wordlist, around 700 words, is an example of this subset.

[8]Constantine [25] makes use of the term *UserRole* with the same meaning.

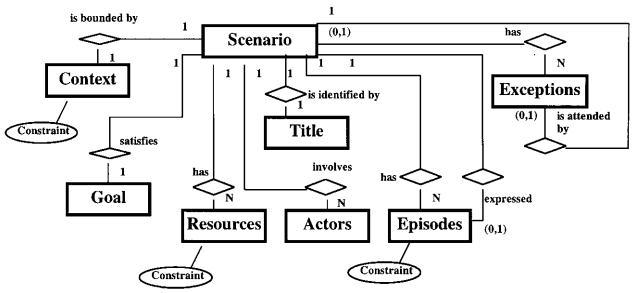


Fig. 4. Entity-relationship diagram for the Scenario Model.

Figure 5 shows the template for describing scenarios based on natural language [2]. The scenario model should be seen as a syntax and structural guidelines to:

- obtain a homogeneous description style among the scenario set;
- demonstrate the several aspects that scenarios can cover; and
- facilitate scenario verification (mainly by an automated process).

Guidance in the structure of use cases and scenarios is shown by Cockburn [35]: 'people in a project are quite confused about how to write them'. That is why Schneider and Winters [24] also proposed a template to describe use cases and scenarios. Jarke et al. [50, p. 162] pointed out that practitioners claimed for guidance in writing structured text scenarios.

A scenario must satisfy a *goal* that is reached by performing its *episodes*. Episodes represent the main course of action but they also include variations or possible alternatives. While performing episodes an *exception* may arise, signalling an obstacle to goal achievement. The treatment of the exception may satisfy the original goal or not.

Although main and alternative courses of action are treated within one scenario, its comprehension is facilitated by the use of natural language, well-bounded situations and the use of *sub-scenarios*. This deals with the scenario explosion problem pointed out by Cockburn [35].

A *sub-scenario* is used when:

- common behaviour is detected in several scenarios;
- a complex conditional or alternative course of action appears in a scenario; and
- the need to enhance a situation with a concrete and precise goal is detected inside a scenario.

The attribute *constraint* is used to characterise non-functional requirements applied to context, resources and episodes.

A scenario may be interrupted by *exceptions*. Each exception is described as a simple sentence that specifies

Scenario: description of a situation in the application domain.
Syntax: Title + Goal + Context + {Resources}$_1^N$ + {Actors}$_1^N$ + {Episodes}$_2^N$ + {Exceptions}

Title: identification of the scenario. In the case of a sub-scenario, the title is the same as the episode sentence (see below in the Episode definition), without the constraints.
Syntax: Phrase | ([Actor | Resource] + Verb + Predicate)

Goal: aim to be reached in the application domain. The scenario describes the achievement of the goal.
Syntax: [Actor | Resource] + Verb + Predicate

Context: composed by at least one of the following sub-components:
Geographical Location: physical set of the scenario.
Temporal Location: time specification for the scenario development.
Precondition: initial state of the scenario.
Syntax: {Geographical Location} + {Temporal Location} + {Precondition}
where Geographical Location is:
Phrase + {Constraint}
where Temporal Location is:
Phrase + {Constraint}
where Precondition is:
[Subject | Actor | Resource] + Verb + Predicate + {Constraint}

Resources: relevant physical elements or information that must be available in the scenario.
Syntax: Name + {Constraint}

Actors: persons, devices or organization structures that have a role in the scenario.
Syntax: Name

Episodes: set of actions that details the scenario and provides its behavior. An episode can also be described as a scenario.
Syntax (using partial BNF):
<episodes> ::= <group series> | <episode series>
<group series> ::= <group> <group> | <non-sequential group> | <group series> <group>
<group> ::= <sequential group> | < non-sequential group>
<sequential group> ::= <basic sentence> | <sequential group> <basic sentence>
<non-sequential group> ::= # <episode series> #
<episode series> ::= <basic sentence> <basic sentence> |
<episode series> <basic sentence>
<basic sentence> ::= <simple sentence> | <conditional sentence> | <optional sentence>
<simple sentence> ::= <episode sentence> CR
<conditional sentence> ::= IF <condition> THEN <episode sentence> CR
<optional sentence> ::= [<episode sentence>] CR
where <episode sentence> is described:
(([Actor | Resource] + Verb + Predicate) | ([Actor | Resource] + [Verb] + Title)) + {Constraint}

Exceptions: usually reflect the lack or malfunction of a necessary resource. An exception hinders the achievement of the scenario goal. The treatment of the exception may be expressed through other scenario.
Syntax: Cause [(Solution)]
where Cause is:
Phrase | ([Subject | Actor | Resource] + Verb + Predicate)
where Solution is:
Title

***Constraint*: a scope or quality requirement referring to a given entity. It is an attribute of Resources, basic Episodes or sub-components of Context.**
Syntax: ([Subject | Actor | Resource] + Must [Not] + Verb + Predicate) | Phrase

+ means composition, {x} means zero or more occurrences of x,
() is used for grouping, | stands for or and [x] denotes that x is optional

Fig. 5. The Scenario Model.

TITLE: Organize the Meeting
GOAL: Assure an efficient development of the meeting.
CONTEXT: The meeting has been previously scheduled.
RESOURCES: equipment, physical space.
ACTORS: requester, secretary.

EPISODES:
The requester instructs the secretary about the meeting call.
CALL TO THE MEETING.
NOTIFY ASSISTANCE.
NOTIFY ABSENCE.
[ASK FOR EQUIPMENT.]
IF the convoking date was made with anticipation THEN REMIND THE MEETING.
[The secretary assures that the equipment is available for the meeting date .]
The secretary assures that the physical space is available for the meeting date . #
EXCEPTIONS:
The requester or main attendees can not attend the meeting. (CANCEL THE MEETING)
The meeting date must be changed. (MOVE THE MEETING DATE)
The need of a meeting requirements change arises. (CHANGE THE MEETING REQUIREMENTS)

Fig. 6. An example of a scenario.

the cause of the interruption. If we include the title of another scenario, the exception will be treated by this scenario.

The *context* is described detailing a geographical location, a temporal location and preconditions. Each of these sub-components may be expressed by one or more simple sentences linked by the logical connectors *and*, *or*.

Episodes are simple, conditional and optional ones. *Simple episodes* are those necessary to complete the scenario. *Conditional episodes* are those whose occurrence depends on a specified condition. The condition may be internal or external to the scenario. Internal conditions may be due to alternative preconditions, actors or resource constraints and previous episodes. *Optional episodes* are those that may or may not take place depending on conditions that cannot be explicitly detailed.

Independently of its type, an episode can be expressed as a single action or can itself be conceived as a scenario, thus enabling the possibility of *decomposition of a scenario in sub-scenarios.*

The scenario model provides the description of behaviours with different temporal orders. A sequence of episodes implies a *precedence order*, but a *non-sequential order* requires the syntax shown in Fig. 5 allowing the grouping of two or more episodes. This is used to express a parallel or indistinct sequential order.

Figure 4 describes the structure of a scenario using the entity-relationship diagram,[9] where only the relevant relationships of entities are shown. Figure 6 exemplifies a scenario of the *Meeting Scheduler* case.

[9]Our actual model is an improvement over the original model presented in Leite et al. [2].

4. Our Process

The general idea of our process is to anchor the scenario description on the vocabulary of the UofD. As such, we start from a pre-existing lexicon in order to build scenarios. The lexicon describes the application vocabulary and the set of scenarios describes the application. The LEL construction process is not detailed here, but we should keep in mind that although the lexicon is build first it evolves as we progress on building scenarios.

We have to understand that the process presented here is as 'it should be' and is a result of four years of experience in building and analysing scenarios. This strategy follows the classical Parnas text on rational design processes [37].

4.1. The Scenario Construction Process

The scenario construction process starts from the application domain lexicon, producing a first version of the scenarios derived exclusively from the LEL. These scenarios are improved using other sources of information and organised in order to obtain a consistent set of scenarios that represents the application domain. During or after these activities, the scenarios are verified and validated with the clients/users to detect discrepancies, errors and omissions (DEO). To describe the process, we will use an SADT[10] [38] model (Fig. 7), where the following activities are mentioned:

1. DERIVE;
2. DESCRIBE;
3. ORGANISE;
4. VERIFY;
5. VALIDATE.

Although the process of scenario construction depicted in Fig. 7 shows a main stream composed of three tasks: DERIVE, DESCRIBE and ORGANISE, these activities are not strictly sequential; some activities may be concurrent due to the feedback mechanism, always present in these situations [39]. There is a feedback when VERIFY and VALIDATE take place, returning to the DESCRIBE activity, where corrections are made based on the DEO lists.[11] The VERIFY activity occurs after describing scenarios and also after organising them when

[10]Notation of SADT: boxes represent activities, left arrows represent input required by the activity, down arrows represent controls, up arrows represent mechanisms and right arrows represents output from the activity.

[11]A DEO list contains the discrepancies, errors and omissions discovered during the validation or verification activities, where suggestions for corrections are included.

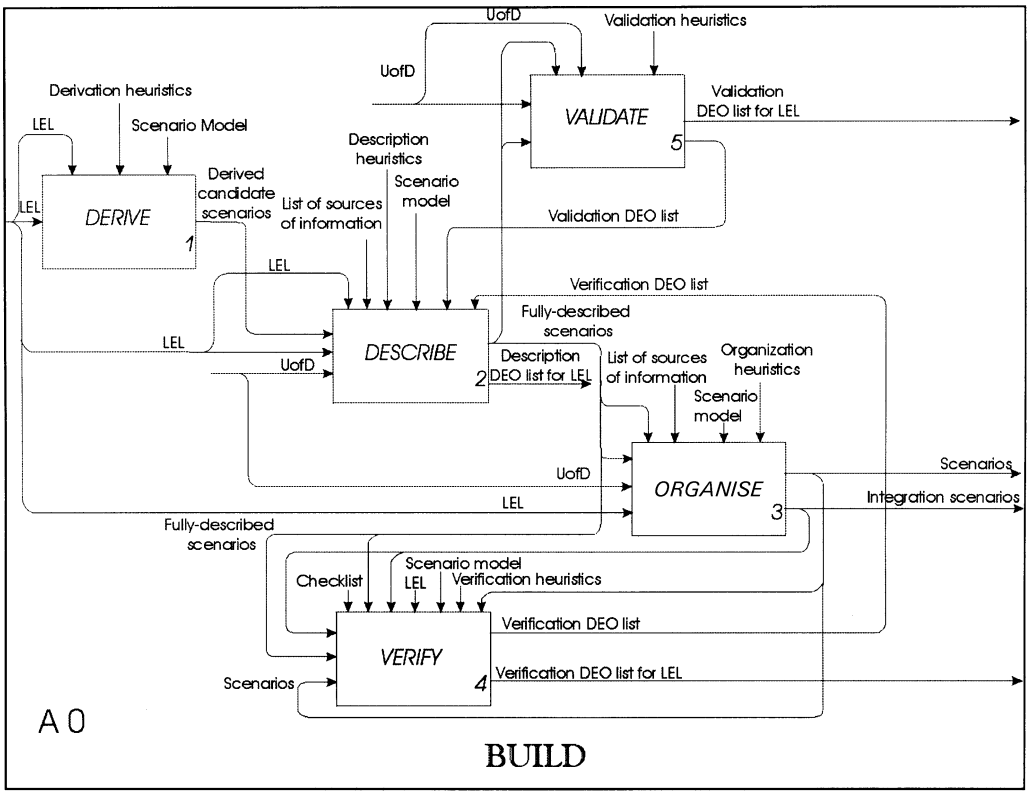


Fig. 7. SADT of the scenario building process.

new scenarios appear or the integration scenarios are generated. Scenarios are validated with the clients/users after verification.

This process might also produce three DEO lists that will act as a feedback for the LEL construction process in order to maintain consistency between the vocabulary of the scenarios and the LEL itself.

4.2. Description of the Activities in the Construction Process

Below we detail the activities in the SADT model of Fig. 7, and for the activities DERIVE and ORGANISE, we detail even further by presenting a SADT model for the decomposition of each one of these activities.

(1) DERIVE

This activity aims at generating the derived candidate scenarios from the information of the LEL using the scenario model and applying the derivation heuristics. The derivation process consists of three steps: identifying the actors of the UofD, identifying candidate scenarios and creating them using the lexicon. Figure 8 shows the SADT model that decomposes the DERIVE activity.

(1.1) IDENTIFY ACTORS

Symbols representing actors of the UofD are identified within the LEL. They should belong to the subject type.[12] Actors are classified into main actors and secondary actors. Main actors are those who execute actions directly in the application domain, while the secondary actors are those who receive and/or give information, but do not share responsibility in the action.

(1.2) IDENTIFY SCENARIOS

The behavioural responses of symbols chosen as main or secondary actors are extracted from the LEL. Each behavioural response represents a possible scenario, and therefore it is incorporated to the candidate scenario list. The scenario title is composed of the action (verb) included in the behavioural response but expressed in infinitive tense plus a predicate also taken from the behavioural response.

When different actors execute the same action, it is very likely that two or more scenarios of the list may share the title; in this case all of them but one should be removed.

[12]LEL symbols are classified into four types: Subject, Object, Verb and State [29].

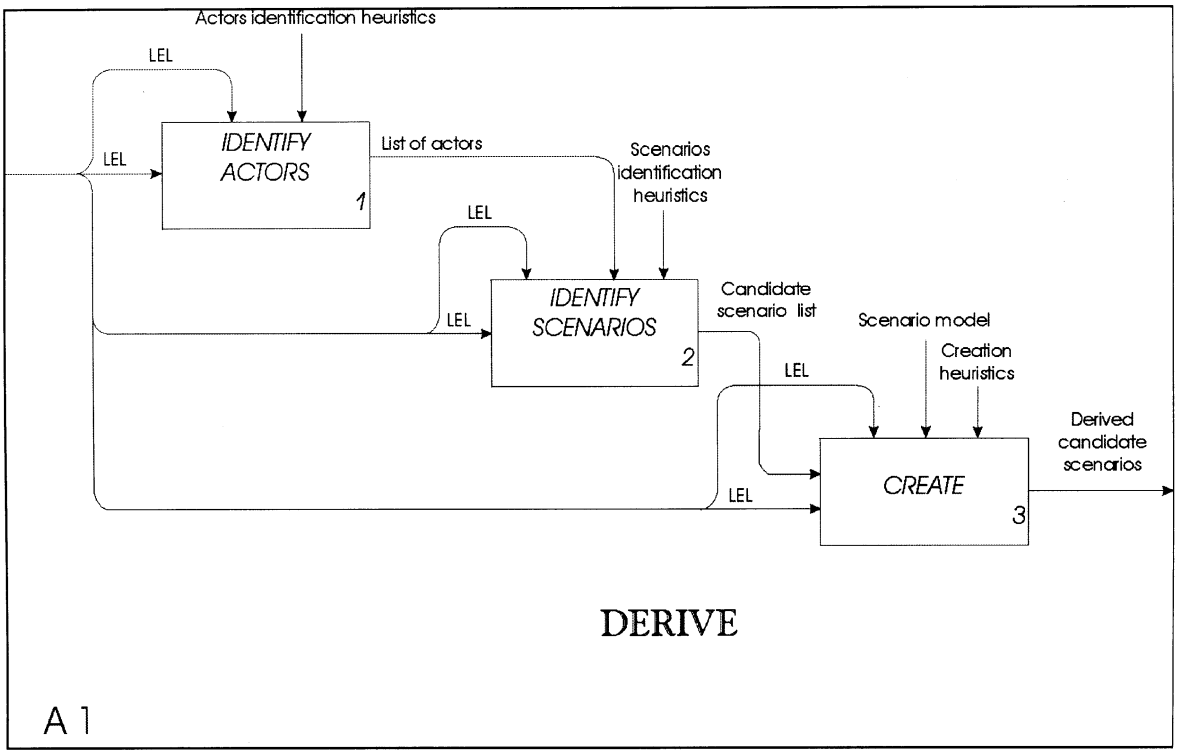


Fig. 8. SADT of the DERIVE activity.

(1.3) CREATE

The intention here is to build the scenario with as much information extracted from the LEL as possible, applying the creation heuristics; the product of this step is the derived candidate scenarios.

The content of each behavioural response that leads to every candidate scenario is analysed in order to find lexicon symbols belonging to the Verb type, from now on called a Verb symbol.

⇒ If a Verb symbol is found within the behavioral response:

The goal is defined based on the title, and the notion of the Verb symbol.

The actors and resources of the scenario are identified from the information contained in the Verb symbol and should also be LEL symbols of the Subject type and the Object type respectively.

The episodes are derived from each behavioural response of the Verb symbol.

⇒ If the behavioural response does not include a Verb symbol:

Lexicon symbols within the behavioural response are identified and considered as a possible source of information.

The goal is defined on the basis of the scenario title.

Possible actors and resources are selected from the above symbols by reading their full definition. Actors are derived from Subject type symbols and resources are derived from Object type symbols.

No episodes are derived from the LEL: they are left for the next step.

In both cases, the geographical and temporal location of the context may be extracted from the behavioural response of the LEL symbol that originated the scenario (Subject symbol). Relevant information that should be registered in the context preconditions may be available not only in that behavioural response, but in other behavioural responses of the same symbol.

Figure 9 exemplifies the DERIVE activity using the *Meeting Scheduler* case and Fig. 10 shows the final version of this scenario. The underlined words or phrases are symbols of the *Meeting Scheduler* lexicon. We will use numbers in Fig. 9 to explain the DERIVE activity: **1** – from the LEL entries we generate a list of actors; **2** – selecting an actor we have the corresponding LEL entry; **3** – from the behavioural response of this LEL entry we produce a list of candidate scenarios; **4** – selecting a candidate scenario we find the LEL Verb symbol corresponding to the scenario title; **5** – we use the notion of LEL Verb symbol to define the goal, actor and resources; **6** – the behavioural response of the

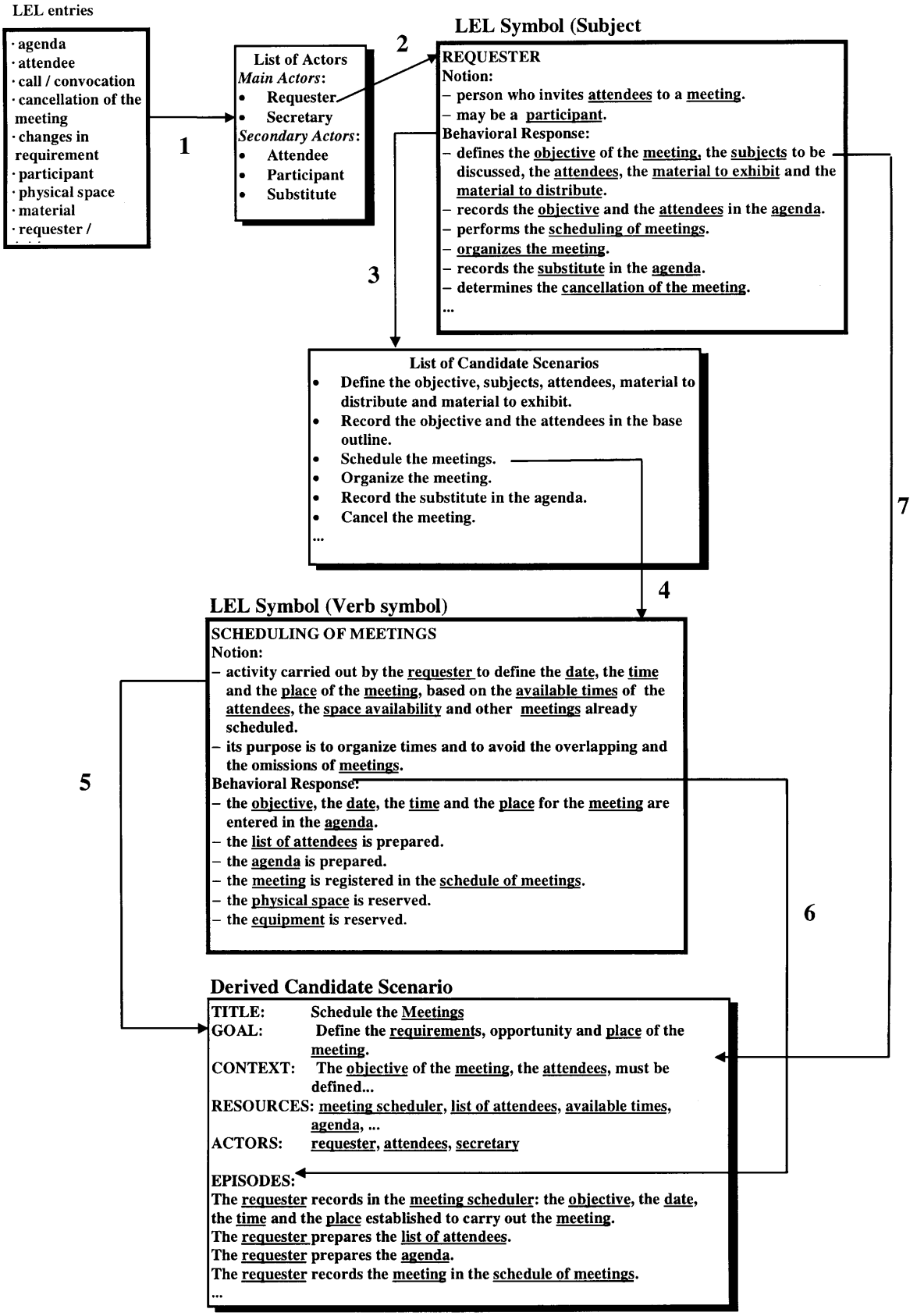
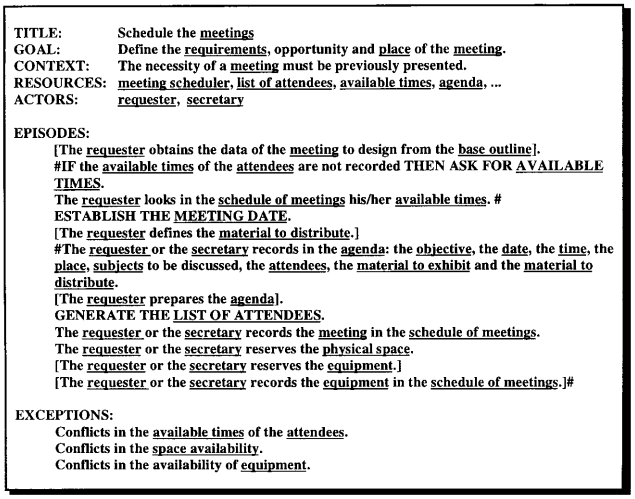


Fig. 9. Graphic example for deriving a scenario.



TITLE: Schedule the meetings
GOAL: Define the requirements, opportunity and place of the meeting.
CONTEXT: The necessity of a meeting must be previously presented.
RESOURCES: meeting scheduler, list of attendees, available times, agenda, ...
ACTORS: requester, secretary

EPISODES:
[The requester obtains the data of the meeting to design from the base outline].
#IF the available times of the attendees are not recorded THEN ASK FOR AVAILABLE TIMES.
The requester looks in the schedule of meetings his/her available times. #
ESTABLISH THE MEETING DATE.
[The requester defines the material to distribute.]
#The requester or the secretary records in the agenda: the objective, the date, the time, the place, subjects to be discussed, the attendees, the material to exhibit and the material to distribute.
[The requester prepares the agenda].
GENERATE THE LIST OF ATTENDEES.
The requester or the secretary records the meeting in the schedule of meetings.
The requester or the secretary reserves the physical space.
[The requester or the secretary reserves the equipment.]
[The requester or the secretary records the equipment in the schedule of meetings.]#

EXCEPTIONS:
Conflicts in the available times of the attendees.
Conflicts in the space availability.
Conflicts in the availability of equipment.

Fig. 10. Final version of a scenario.

LEL Verb symbol will provide the basis to describe the episodes; and **7** – from the behavioural response of the LEL originated in **3** (Subject type) we derive the context information.

(2) DESCRIBE
This activity aims at improving the candidate scenarios by adding information from the UofD using the scenario model, the lexicon symbols in the descriptions and applying the description heuristics. The result is a set of fully described scenarios.

After the previous step, an incomplete set of candidate scenarios is obtained. They require to be extended in two senses: new scenarios might be added and the scenario content should be improved. Thus, if available, the information gathered during the LEL construction process, but not included in the lexicon, is used here as well. When needed, the requirements engineers return to the sources in order to collect more information.

This activity should be planned and usually relies on structured interviews, observations and document reading. First, information gathering aims at confirming and improving normal courses of events. Afterwards, alternative and exception cases should be detected. Some causes of exception are elicited from the sources of information while others may be deducted by questioning the occurrence of episodes or the unavailability or malfunction of resources. When discovering causes of exception, the requirements engineers should inquire how the exception is treated in the UofD; this implies a new situation that might need to be described through a separate scenario. Constraints may also be detected, some elicited from the UofD and others by examining the episodes. After finishing the scenario set description, episodes should be carefully reviewed to confirm sequential or non-sequential ordering and to detect optional episodes.

Observing Fig. 7, we can see that there is a feedback from the verification and validation processes, here represented by the DEO lists (lists of Discrepancies, Errors and Omissions). These lists may motivate changes in the scenario description. It is also the case that, in the DESCRIBE activity, we may generate a DEO list for the LEL.

Below, we list some general heuristics regarding this activity. Since some scenarios might already be partially described at this point, these guidelines should be used to review initial descriptions.

- Short sentences are preferred.
- The sentences should be written maximising the use of the lexicon symbols.
- The use of more than one verb per sentence should be avoided.
- Actors and Resources should be preferentially lexicon symbols.
- The goal must be precise and concrete.
- At least one of the sub-components of the context must be filled.
- The component Resources should list those involved in the episodes or implicitly referred to the episode verb, excluding trivial resources.
- The component Resources should not include those needed in sub-scenarios.
- The component Actors should list those involved in the episodes.
- The component Actors should not include those needed in sub-scenarios.
- The verb of each episode should be precise and concrete, specifying the final action and avoiding ambiguity and vagueness.
- Every episode must be enacted within the geographical and temporal location described by the scenario Context.
- The present tense should be preferred in describing episodes.
- Avoid using ‘should’, ‘can’, ‘may’, ‘must’ in episodes.

(3) ORGANISE
This activity is the most complex and more systematised one in our scenario construction process. Its root is the idea of *integration scenarios*, ‘artificial’ descriptions with the sole purpose of making the set of scenarios more understandable and manageable. Integration scenarios give a global vision of the application.

When facing large applications, the number of scenarios could be unmanageable and the requirements engineer becomes sunk in details, losing the global

vision of the application. In order to face this problem, we propose the construction of integration scenarios, which give an overview of the relationship among several situations, since each integration scenario episode corresponds to a scenario. Therefore, the main purpose of integration scenarios is to link disperse scenarios providing a global vision of the application, while preserving the natural language format encouraged in our approach.

Thus, scenarios are organised by reorganisation and integration, beginning with the fully described scenarios. Although ORGANISE is located as third box at Fig. 7, the activities VERIFY and VALIDATE must occur before we start the organisation of fully described scenarios. In other words, organising scenarios is a task that is done only when there is a good confidence in having a well-defined scenario set, ideally when the DEO lists are empty.

In this activity we may need to go back to the UofD, using the LEL and the scenario model and applying the organisation heuristics. Nonetheless, the fully described scenarios may still have some important weakness, such as:

- lack of homogeneity;
- minor semantic problems; and
- lack of global perspective.

We understand that these problems may be minimised by the heuristics that we detail for the ORGANISE activity.

Figure 11 shows the SADT model that decomposes the ORGANISE activity. The heuristics used in this activity reflect our experience on building scenarios. We also have produced taxonomies of operations and relationships that are used by these heuristics.

(3.1) REORGANISE

Scenarios may have different degrees of detail or overlapping information, especially when built by more than one requirements engineer. This situation gets more complex as we have more scenarios and a larger team. Basically reorganising scenarios consists in putting together two or more scenarios in one or breaking a scenario in two or more. Putting together is done when a unique scenario becomes artificially divided during the DERIVE or DESCRIBE activities. On the other hand, we break a scenario when it contains more than one situation. We describe these operations as pairs in different dimensions. The criteria to decide which operation to use in each dimension are given by heuristics. The three main dimensions in which scenario reorganisation can be applied are:

- sub-scenario;
- situations with variations;
- temporal context.

During the DERIVE and DESCRIBE activities focus is not on the sub-scenario; thus at this moment derived sub-scenarios require special attention to confirm that they are needed and any hidden sub-scenarios should be found to clarify the scenario set.

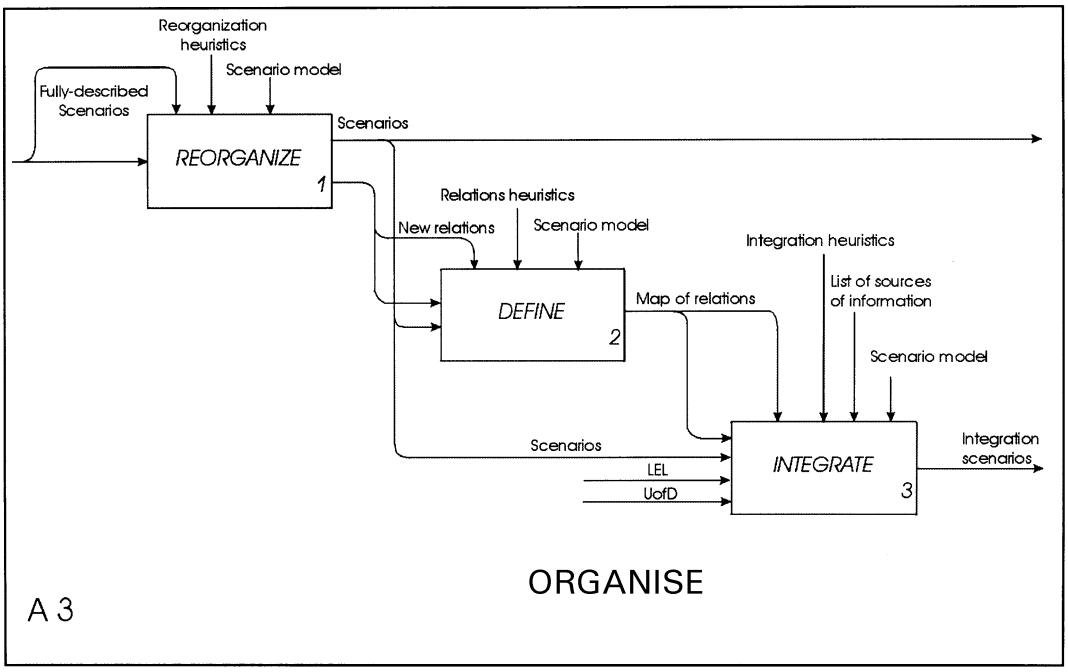


Fig. 11. SADT of the ORGANISE activity.

In the UofD there may be situations very similar to each other. How they were elicited for the scenario set depends on several factors, such as the number of requirements engineers involved, the order in which the behavioural responses of LEL Subjects were considered, and the viewpoints taken in consideration.

The duration of a scenario is also a difficult matter to establish. In some cases two scenarios are so related that one is produced immediately after the other; thus they may be better described as just one scenario. On the other hand, a scenario with a long life span (temporal context) containing episodes that are not very tight semantically may lead to the creation of two or more different scenarios.

Below we define, for each dimension in which scenario reorganisation can be applied, the operations that should be used for each dimension.

There are two operations in the sub-scenario dimension:

- Flatten
- Factor

There are two operations in the variation dimension:

- Consolidate
- Divide

There are two operations in the temporal context dimension:

- Fusion
- Split

In order to establish the necessity of composition/ decomposition operations, some scenario properties and relationships among them should be previously identified. When a scenario presents special properties mainly in its episodes, decomposition may be applied. On the other hand, composition may be used when discovering specific relationships among scenarios: strong overlap, precedence order or hierarchy relation.

It should be noticed that either the composition or decomposition operation involves the risk of semantic degradation. Decomposition creates two or more scenarios where there was previously just one. This procedure must not create ‘artificial’ situations; in other words, decomposition must be applied only when the resulting scenarios better describe the UofD. On the contrary, composition replaces two or more scenarios by one.

Operations are suggested as guidelines to improve scenario comprehension and management. Therefore, reorganisation should not be seen as a mandatory activity but as good practice.

Below, operations with their corresponding properties or scenario relationships are described:

- *Flatten:*[13] May be applied when detecting non-relevant sub-scenarios with few occurrences in other scenarios. Flatten incorporates the episodes of the sub-scenario inside each scenario that mentions it. The sub-scenario should present the same granularity of every flattened scenario. The original sub-scenario is deleted when all occurrences are flattened. This operation allows to reduce the number of non-relevant scenarios and, thus, to ease their management. The resulting scenario set decreases the hierarchy depth in all flattening points. This operation can also be seen as the inline feature of code generation in some program languages.
- *Factor:* May be applied when a set of very relevant episodes or a set of episodes with different level of detail in relation to the rest is detected; in either case the set deserves special treatment. Also, when discovering the occurrence of the same set of episodes in two or more scenarios having common behaviours this operation may be used. Factor creates a sub-scenario factoring episodes from one or more scenarios. The group of episodes is replaced by the title of the sub-scenario holding them. Factor makes scenarios easier to understand and more reusable.
- *Consolidate:* May be applied to overlapping scenarios. Two or more scenarios strongly overlap if their goals and contexts are similar and they have several coincidences in episodes. The original scenarios must present the same ‘core of action’ [14]. Consolidate copies the common episodes of the original scenarios to the new one and creates new conditional episodes using the non-shared episodes, the condition being the corresponding part of the original scenario precondition. Occasionally new symbols may be added to the LEL. Original scenarios are deleted. In consequence, this operation allows reducing redundancy in the scenario set.
- *Divide:* May be applied when the presence of several conditional episodes, driven by the same condition, is detected. Divide produces two new scenarios, moving the trigger condition to both preconditions. Non-trigger condition-based episodes are put in both scenarios. On the other hand, trigger condition-based episodes are moved to their corresponding scenario with the removed condition. The original scenario is deleted. This operation helps to avoid cluttered scenarios, thus reducing complexity.
- *Fusion:* May be applied to scenarios that present a contiguous precedence order, compounded goals and coupled contexts. No temporal gap may exist among

[13] In [14], another scenario operation, called Encapsulate, was defined and used; actually flatten and fusion are a specialisation of Encapsulate.

these scenarios. A temporal gap occurs when there is a long time interval between two scenarios. Fusion copies the episodes of every original scenario to the new scenario in the corresponding order. The original scenarios are deleted, giving an opportunity to reduce the number of scenarios and, thus, to ease their management.

- *Split:* May be applied to a scenario if there is a temporal gap between episodes or when detecting a very extended temporal context. An extended temporal context may be detected straightforwardly by the context itself or by the sequence of the episodes. Also, discovering more than one geographical location in a scenario could be used as a guide for considering the necessity of a split operation. Therefore, the scenario is representing more than one situation, contradicting the scenario definition. The decomposition of the scenario should produce scenarios with a well-bounded temporal context. Split copies all the episodes that precede a temporal gap to a new scenario and those following the temporal gap to another new scenario. The original scenario is then deleted. Preconditions of the second and following new scenarios may reflect the contiguous precedence order established. Thus, split allows to reduce complexity and to better understand situations.

Figure 12 shows the factor operation applied to two scenarios with several common episodes that have a specific goal; a new scenario holding those episodes is created and the original scenarios are rewritten, replacing the common episodes by the new scenario title. Figure 13 exemplifies the consolidate operation applied to two overlapped scenarios, producing a single scenario holding common episodes while non-common episodes are incorporated into a conditional statement.

When applying composition operations, the goal of the resulting scenario should be extended to embrace the whole situation, especially during fusion; in case of consolidate the goal is extended by generalisation. On the contrary, when applying decomposition, the goal of the new scenario may be less global than the original one.

Since consolidate and divide are opposite operations, after a consolidation the resulting scenario may be considered for a division returning to the original state and vice versa. Criteria to decide to go back or forth are required. Consolidate should be applied if few conditional episodes appear in the resulting scenario and should be avoided otherwise. On the other hand, divide should be chosen only if several trigger-condition-based episodes are affected by the operation.

Fusion and split are opposite operations too; thus both require criteria to decide the best direction to go. The keyword here is the temporal gap, either with respect to scenarios or to episodes. A delay between two episodes may be considered as a temporal gap only if it cannot be embraced by the scenario temporal context. In other words, a specific episode could be preceded by a temporal gap in one scenario but not in another one. It then becomes clear when to split or fusion scenarios.

The sub-scenario dimension operations also require criteria to avoid looping from one state to the opposite. Therefore, flatten should be applied when flattened episodes in the resulting scenario have the same level of detail as the rest of the episodes and when the original sub-scenario does not represent a situation by itself, which could be confirmed by a lack of meaning of its goal. On the contrary, factor should be used only if a meaningful goal can be found for the new sub-scenario as a consequence of having discovered a specific and actual situation inside a larger scenario.

(3.2) DEFINE

Several authors have proposed scenario relationships, such as Jacobson's uses association and extend associations [27], Booch's semantic connections [16], Dano's temporal links [20] and others. Those relationships were identified and found valuable for other purposes. Since the target of the relationships that we are going to define here is the construction of integration scenarios, a new set of relations is necessary. As such the DEFINE activity identifies different relationships among scenarios in order to be able to integrate them.

Below we detail each type of relationship we have identified in our work with scenario integration.

- *Hierarchical relationship* is the one established between scenarios and sub-scenarios. This relationship comes naturally while describing the scenarios or reorganising them. The behaviour of a scenario is described through a set of episodes, which could be simple actions or sub-scenarios; the latter case occurs when the action is more complex or is common to several scenarios. A scenario may contain more than one sub-scenario or none. A sub-scenario could be included in one or more scenarios and can itself contain sub-scenarios, allowing many levels of depth in the hierarchy. Thus, we can define a hierarchy as a set composed of a scenario and its sub-scenarios.
- *Overlap relationship* is that one established among scenarios with common portions. This relationship is observed mainly when several common episodes are present among different scenarios; thus common actors and probably common resources appear.
- *Order relationship* is the one established between two hierarchies when one precedes the other. A hierarchy may come before other hierarchies setting up a partial temporal ordering with them. Thus, a hierarchy may have zero or more predecessors and zero or more

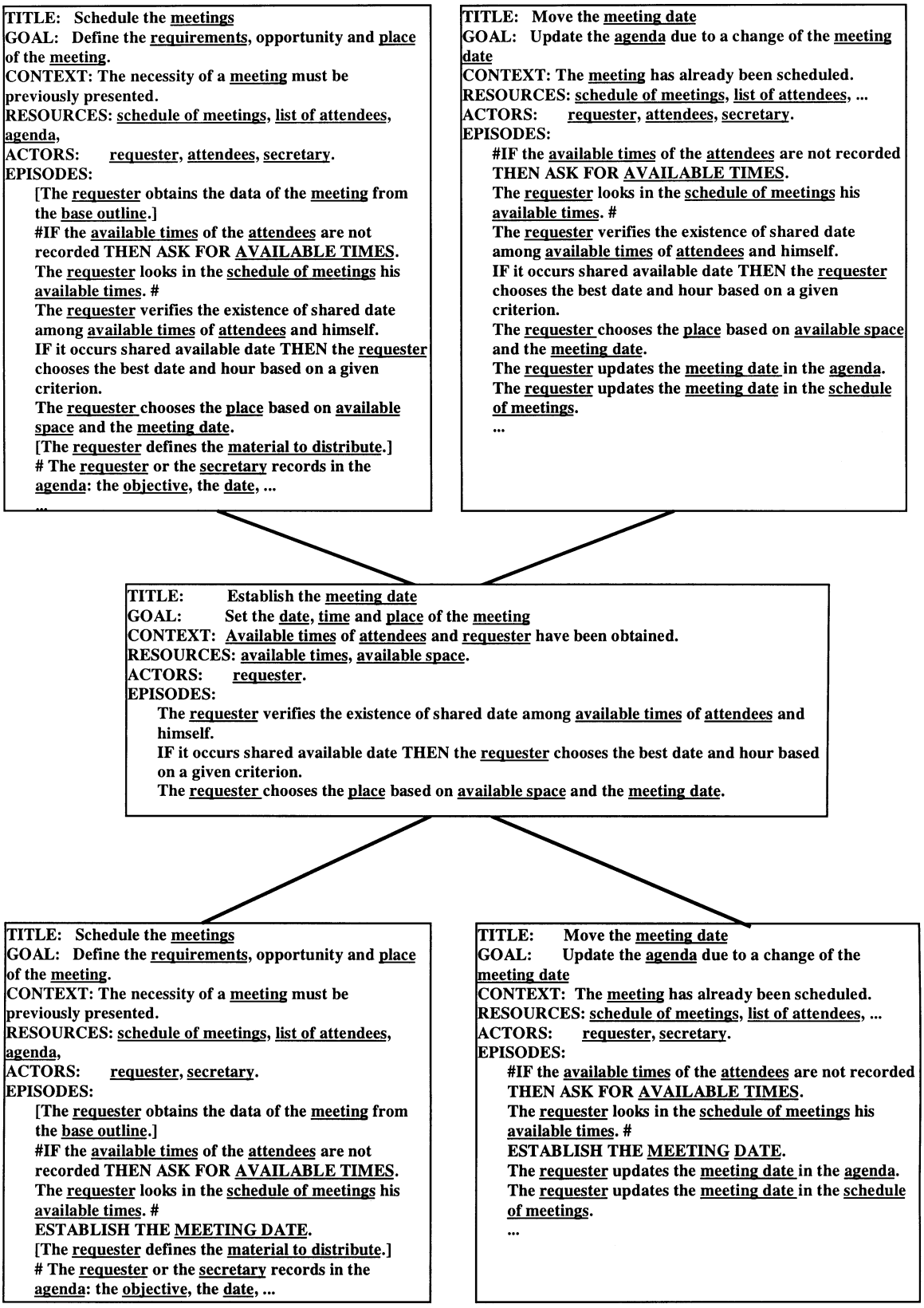


Fig. 12. An example of factor.

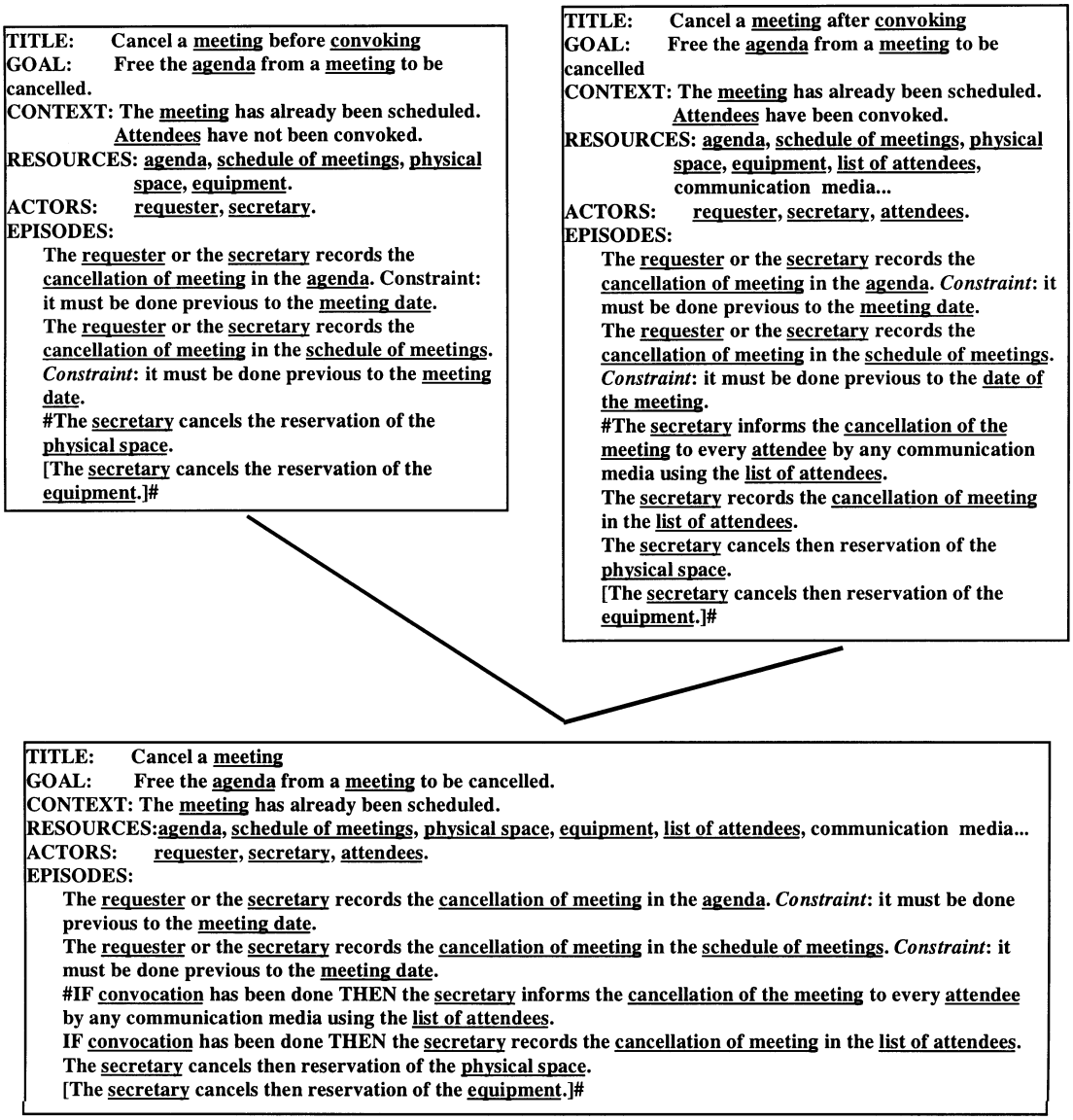


Fig. 13. An example of consolidate.

successors. A sequence is established when a hierarchy is immediately preceded by another one. Since the second may be also preceded by a third one, a large number of hierarchies may be involved in a sequence beginning with at least one leader hierarchy. It should be noticed that a sequence could have more than one leader hierarchy and also more than one trailer hierarchy.

- *Exception relationship* is the one established between a scenario and those scenarios that treat exceptions. When a scenario has an exception, its cause is described and if a treatment of the exception is specified a scenario for treating the exception is mentioned. A scenario may be related to one or more exception scenarios. An exception scenario may treat exceptions occurring in different scenarios.

If two scenarios are linked by an exception relationship they will be put together in one integration scenario as a sequence, where the normal scenario comes first, followed by a conditional episode holding the exception condition and the solving exception scenario.

The overlap relationship is not currently used in the scenario integration activity since it is essentially based on hierarchy and order relationship. However, the possibility of integration by generalisation has been detected. The scope, advantages and inconveniences of integration by generalisation are still being studied.

(3.3) INTEGRATE

The INTEGRATE activity implements the middle-out behaviour of our construction process. Its first three steps are bottom-up and the last two steps are top-down. As such the integration of scenarios could be seen as a five-step procedure. The four initial steps mainly rely on syntax information, but the last one requires more semantic knowledge. In this activity, first, scenarios are grouped in hierarchies and then hierarchies are grouped in sequences. Finally these sequences are used to build integration scenarios. Figure 14 shows an example of an integration scenario produced for the *Meeting Scheduler* case. Note that since an integration scenario is produced just to organise a set of scenarios, it does not use the entities: resources and actors.

Build Scenario Hierarchies is the first step of the scenario integration activity. A hierarchy is a set of scenarios linked by hierarchical relation. Every hierarchy has a root scenario, which is not referred to as sub-scenario by another scenario. The hierarchies are identified by their root scenario. Isolated scenarios are hierarchies by themselves.

Detect Partial Order among Hierarchies is done based on the comparison between the preconditions or resources or constraints of a hierarchy against the title, goal or episodes of other hierarchies. If a precondition of a hierarchy identifies an initial state that is satisfied by another hierarchy there is a partial order between this two hierarchies. The same may occur with a resource needed in one hierarchy and produced by another one. Besides, episode and resource constraints may be satisfied by another hierarchy, thus establishing a partial order. These constraints can be used for comparison when they point out to resource states or certain conditions that should be previously met. Preconditions, resources and constraints of hierarchies are not obtained from the root scenario itself but from the whole hierarchy. To establish hierarchy preconditions, first preconditions of all the scenarios belonging to the hierarchy must be unified; then, preconditions required by a sub-scenario and satisfied by another sub-scenario must be removed from the union. The resources and constraints of the hierarchy are obtained in the same way. The goal of a hierarchy is just the root scenario goal and the comparison should stop as soon as a partial ordering is found or detected as impossible. If the partial order between two hierarchies only takes place under certain conditions, it is labelled with the condition. To reduce the search for hierarchy providing resources or satisfying constraints and preconditions it is better to check against title and goal in the first place and check against episodes only on failure. If a comparison is successful, partial ordering is found, otherwise the next comparison takes place. If no comparison is successful, then there is no partial ordering. The comparisons used in this step are:

- hierarchy precondition against titles of other hierarchies;
- hierarchy constraints against titles of other hierarchies;
- hierarchy resources against titles of other hierarchies;
- hierarchy precondition against goals of other hierarchies;
- hierarchy constraints against goals of other hierarchies;
- hierarchy resources against goals of other hierarchies;
- hierarchy precondition against episodes of other hierarchies;
- hierarchy constraints against episodes of other hierarchies;
- hierarchy resources against episodes of other hierarchies.

Build Hierarchy Sequences begins with the review of the partial ordering relationship detected among hierarchies. This is done to clarify which is the set of hierarchies that immediately precedes a given one. To do this, all partial orders derived by transitivity are deleted. It is said that among hierarchies A, B and C there is a transitivity if A and B precedes C but also A precedes B. Transitivity is not easy to remove. However, it can be eliminated by grouping all partial order relations with the same hierarchy on the left side of the relation. Any additional partial order between hierarchies on the right side of a relation is used to identify a transitivity. New information from the UofD may be elicited when gaps between hierarchies are found. This happens when two hierarchies in sequence do no match properly since one or both lack some actions that actually take place in the UofD. Isolated hierarchies are sequences by themselves.

Build the Integration Skeleton is done following the sequences obtained in the previous step. This step builds only the episode components of the integration scenarios. First a main scenario is built and then intermediate-

TITLE: Manage the Meeting Scheduler
GOAL: Manage the Meeting Scheduler efficiently and effectively.
CONTEXT: There is a subject to treat or resolve by two or more persons.
RESOURCES: -
ACTORS: -

EPISODES:
DEMAND A MEETING.
SCHEDULE THE MEETINGS.
ORGANIZE THE MEETING.
IF requester or main attendees can not attend the meeting THEN CANCEL THE MEETING.
IF the meeting date must be changed THEN MOVE THE MEETING DATE.
IF the need of a meeting requirements change arises THEN CHANGE THE MEETING REQUIREMENTS. #
IF the meeting was not canceled THEN ATTEND THE MEETING.

Fig. 14. An example of an integration scenario.

level integration scenarios may be obtained. All sequences are put in the main integration scenario marked with the non-sequential indicator (#). If a sequence is composed of more than one scenario, a stub for an intermediate-level scenario is created and a nominal title is used in both the main integration scenario and the stub. This is applied to every existing stub creating new stubs when non-sequences of subsequences show up.

Compose Title, Goal and Context for the Integration Scenarios is the step where new semantic information needs to be added. This is done by watching the scenarios involved in the integration scenario. This step acts as a sort of final verification, since the title and the goal should be written preserving cohesion; in other words, no conjunctions should be allowed. Descriptions should be written based on the LEL.

(4) VERIFY

The VERIFY activity is performed at least twice during the scenario-building process, the first one over the fully described scenario set and the second after the ORGANISE activity. This is done following a checklist with verification heuristics. As a consequence of this activity, two DEO lists are produced: one used at DESCRIBE and the other used during the LEL construction process. The verification is divided into intra-scenarios, inter-scenarios and against the LEL.

The *intra-scenario verification* includes:

- Verify syntax:
  - Check the completeness of every component.
  - Check the existence of more than one Episode per scenario.
  - Check the syntax of each component as established in the scenario model.
- Verify relationship among components:
  - Check that every Actor participates in at least one episode.
  - Check that every Actor mentioned in episodes is included in the Actor component.
  - Check that every Resource is used in at least one Episode.
  - Check that every Resource mentioned in Episodes is included in the Resources component.
- Verify semantics:
  - Check coherence between the Title and the Goal.
  - Ensure that the set of Episodes satisfies the Goal and is within the context.
  - Ensure that actions present in the preconditions are already performed.
  - Ensure that Episodes contain only actions to be performed.
  - Ensure the same level of detail inside a scenario.

The *inter-scenarios verification* includes:

- Verify sub-scenarios existence:
  - Check that every Episode identified as a sub-scenario exists within the set of scenarios.
  - Check that the set of Episodes of every sub-scenario is not already included in another scenario.
  - Check that every exception is treated by a scenario.
- Verify context:
  - Check that every precondition is either an uncontrollable fact or is satisfied by another scenario.
  - Check coherence between sub-scenario preconditions and scenario preconditions.
  - Check that geographical and temporal locations of sub-scenarios are equal or more restricted than those of scenarios.
- Verify equivalence in scenario set:
  - Check that Goal coincidence only takes place in different situations.
  - Check that Episode coincidence only takes place in different situations.
  - Check that Context coincidence only takes place in different situations.

The *against LEL verification* includes:

- Check that every lexicon symbol is identified.
- Check the correct use of lexicon symbols.
- Check that Actors are preferentially Subject symbols.
- Check that Resources are preferentially Object symbols.
- Check that the behavioural responses of Subject symbols are covered by scenarios.

Using the verification DEO lists, the scenarios and the LEL are modified. If major corrections were needed, a new verification could be required. When the Describe–Verify cycle is over, the ORGANISE activity should provide a larger set of scenarios, which in turn are verified, starting a possible Describe–Organise–Verify cycle. This activity is actually done by means of an inspection process [40].

Its is important to stress that our verification process is driven by syntax checks, by cross-referencing the LEL and the scenarios and by a checklist. Some of the heuristics of this checklist were given above. The checklist registers what was learned in terms of verification during our experience with scenarios.

(5) VALIDATE

Scenarios are validated with the clients/users usually by performing structured interviews or meetings. During

validation, the doubt component must be considered with special attention for those scenarios still presenting it.

It is important to stress that, although using a structured description, our scenarios are written in natural language, employing their client/user's own vocabulary and describing a specific well-bounded situation.

The validation task allows the detection of DEO in scenarios. Errors are mainly detected by reading each scenario to the client/user; some omissions may also appear during that reading but others by questioning the lack of information or details in scenarios. Discrepancies may appear during interviews but they mainly arise afterwards, when analysing the collected information.

Note that the primary goal of validation is to confirm the elicited information and to detect DEO; however, a side effect of this process is the elicitation of new information. The validation of a set of scenarios, after a verification process, should confirm that the Universe of Discourse situations, occurrences, have been reported in accordance with the perception of the UofD actors (clients/users) in charge of reading and discussing the scenarios.

4.3. Observations from the Meeting Scheduler Case

The box below summarises the evolution of the set of scenarios during the construction process for the *Meeting Scheduler* and Fig. 15 shows graphically the numbers mentioned inside it.

Evolution of scenarios:
- 23 scenarios in the initial list
- 16 scenarios in the final list
- 14 scenarios were kept from the initial list to the final list
- 9 scenarios of the initial list were included as simple episodes inside other scenarios
- 2 scenarios of the final list were generated after the verification and validation processes (these two scenarios were previously included as simple episodes inside other scenarios)
- one integration scenario
- 87% of the scenarios were obtained by applying the derivation heuristics.

The composition of the 16 final scenarios was:
- 4 scenarios
- 9 sub-scenarios
- 3 exceptions

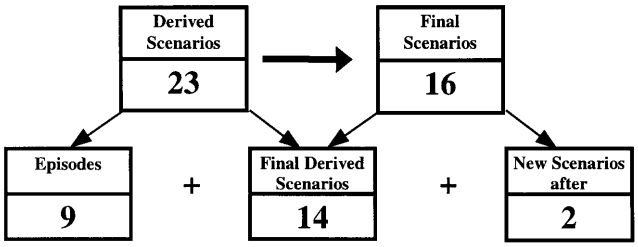


Fig. 15. Result of the organisation process for the Meeting Scheduler.

5. Using the process

The process described here, as stated before, is the last polished version of the process which has been evolving since the first case study in 1996 and has included the lessons learned during the construction and analysis of more than 400 scenarios. Here we give a brief description of some of these case studies. Most of them were done by third parties in a university environment (graduate and undergraduate seminars) at the universities involved in this project as well as by the researchers themselves. Some of them were checked with the potential clients, and at least one of them has resulted in a real system (PUC-Rio Informatics Department graduate advancement control system). We have analysed each of these sets of scenarios to come up with the construction process just presented. Some of the case studies did go through an organisation process (based on the process we present here), but others did not. We list below the problem classes where scenarios were developed.

1. *Argentine passports issue system* [2,42]. A set of 24 scenarios and 34 LEL symbols were produced by a team of researchers. This system is centralised in the Federal Police Department, which is the unique organisation authorised for the issue of passports for the whole country. The complexity of the system comes up from the strict control that is necessary and the high volume of information that it deals with.
2. *Meeting scheduler system* [41,43]. Three different sets of scenarios (16,13 and 17), using an LEL of 34 entries, were generated, two by undergraduate student groups and one by a team of researchers. The Universidad de Belgrano was used as the target organisation based on the case study proposed by van Lamsweerde [31].
3. *Saving plan for automobile acquisition system* [44, 45]. Nine different sets of scenarios were generated, seven by undergraduate student groups and two by teams of faculty members. The goal of this system is to manage saving plans for the acquisition of brand

new vehicles. A group of physical or legal persons is constituted and participates monthly in an adjudication process organised by a general manager in order to deliver a vehicle. An average of 18 scenarios for each group were produced, ranging from 8 to 22 scenarios. The LEL varied from 16 entries to 68 entries.

4. *Graduate advancement system* [14]. Five different sets of scenarios were generated by graduate student groups. From one of these five sets, a real system was modelled and developed for the Informatics Department of PUC-Rio and it manages all operations regarding the control of graduated students (http://www.inf.puc-rio.br/~dilbert).
5. *Library system*. Four different sets of scenarios were generated by groups of senior undergraduate students. The system consists basically of recording books and users and checking the loan of books.
6. *Text editor system*. Eight different sets of scenarios developed by senior undergraduate students.
7. *The lexicon and scenarios for the lexicon and the scenario construction processes* [46]. A set of 64 scenarios and 78 LEL entries were produced by an undergraduate student group and two researchers. This lexicon describes all the important items in the construction of the lexicon and scenarios, following our approach. The set of scenarios describes the situations that occur during the construction of the lexicon and the scenarios. Both the lexicon and scenarios were analysed by the authors of the present paper. The lexicon and scenarios are available at (http://usuarios.arnet.com.ar/ogarcia).

Some of the above sets were developed intuitively from scratch, beginning with the UofD and following the idea of finding situations, while others were constructed using an approach similar to the one previously described. In the case studies performed without following our process, scenario producers noticed the necessity of having an overview of the problem, materialised in one or more 'general scenarios' [41] or another model that would help to get a first idea of the application domain. This idea is similar to Booch's primary scenarios [16] and to Sutcliffe's grounding scenarios [18]; that is, the general scenario makes it possible to have an abstract view of the set of scenarios.

On the other hand, when the scenario set was developed following the heuristics described in the previous section, there was no need for a general overview at the beginning of the process, since we were not using a top-down approach. Our process answers the question of how to organise scenarios, when we have many of them. As such, we believe that the process reported here has a clear-cut distinction between creating the scenarios for the several situations and then organising them as a model, in the software engineering sense.

6. Conclusions

Scenarios have been attracting much attention from the RE community. A significant number of articles have been dealing with the subject and there is a continuous effort to improve processes and scenario representations. We consider our work a contribution mainly to the area of scenario management [50].

From other work [8, 47] we became convinced of the importance that scenarios have for describing the behaviour of objects and the advantage of tracing this behaviour through the software system interface. From Carroll [5] we achieved a better comprehension of the cognitive aspects of scenario-based development and also a confirmation that scenarios must begin in the Universe of Discourse and not only in the software system interface. This idea was also reaffirmed by Zorman [9], who well defines scenarios as situations; in addition, we utilise her study of scenario representation. Potts [15] showed us the relevance of relating scenarios with goals and thus proved important our previous knowledge of the meta model oriented to goals proposed by Dardenne et al. [48]. In particular, we differ from Potts [15] because we use a scenario hierarchy instead of a goal hierarchy and our construction of hierarchies uses a bottom-up strategy against Potts's decomposition of goals.

Dano et al. [20] consider that *Use Cases* and scenarios are interchangeable because they capture specific ways of using the system functionality, while for us scenarios capture situations of the application domain. Besides, [20] proposes the utilisation of formalisms as an alternative to natural language. Our approach consists of using natural language with rules for eliciting and then modelling scenarios. We agree with Rolland [12] that the use of natural language in scenarios implies an easy-to-understand model though it may be attached with ambiguities and inconsistencies. These drawbacks can be reduced by employing a well-defined UofD vocabulary such as our LEL [29], complemented with a verification process such as presented here. With respect to Sutcliffe [18], we share entirely the convenience of integrating methods for the requirements production. Nonetheless, the idea of developing a prototype that acts as 'concept demonstrator', as proposed by that author, only helps to establish the requirements concerning the man–machine interface and that are specific to the artefact in construction, leaving out other aspects of the application domain.

The operations presented in the ORGANISE activity (section 4) were inspired by Breitman's work [14], which discovered, from analysis of sets of scenarios, that scenario authors have used certain types of operations in producing the scenarios. The difference is that our operations were designed to achieve the goal of producing an organised set of scenarios.

During the learning process that we went through, it became clear to us that we have deployed two different ways of organising scenarios: one that we called general scenarios and the other which we called integration scenarios. General scenarios are the root scenarios that we use when following a top-down strategy. Table 1 synthesises the characteristics of using general scenarios versus integration scenarios, learned through our experience in the case studies already mentioned.

In a top-down approach the general scenarios are described at the beginning of the scenario construction process, from poor and general application domain knowledge. Conversely, the integration scenarios in a middle-out approach are described as the last step of the process, after acquiring large and detailed domain knowledge.

General scenarios need continuous maintenance along the construction process in order to achieve consistency with the rest of the scenarios, otherwise they will lose their utility and should be put aside. As the integration scenarios are built at the end of the process, they are naturally consistent with the rest of the scenarios.

General scenarios give an approximate image of the application domain at the beginning of the process and with a great maintenance effort they could provide a view similar to the integration scenarios. This effort is directly related to the possibility of verifying the consistency among scenarios. The integration scenarios episodes show the relationships among scenarios, while the general scenarios show activities that may or may not be scenarios.

In practice [11], general scenarios are used to divide tasks, not yet well known, among the team of engineers. This is neither possible nor desirable with our approach.

Table 1. Comparative table for scenario construction

Characteristics	General scenario	Integration scenario
Approach	Top-down	Middle-out
Construction	At the beginning	Close to the end
Refinement need	Yes	No
Consistency effort	High	Very low
Full application domain view	Approximated	Proximal
Inter-scenario view	Poor	Clear
Allow problem division	Yes	No
Allow consistency	Only if refined	Yes

As pointed out by Jackson (see section 2), a top-down approach is useful to organise a problem when it is previously known but not before.

Based on our experience with scenarios we have concluded that there is an advantage of using integration scenarios instead of a top-down organisation. We have stated that our 'middle-out' strategy produces a set of scenarios that, although, representing different UofD situations, are organised following software engineering modelling principles We consider that our contribution to scenario research is to unveil some of the aspects of the scenario construction process. Although our work is biased by our own representation model (Fig. 5), we believe that the aspect of composition/decomposition is of general interest to scenario construction. We see the theme of composition/decomposition related to the abstraction facet (Contents View) of the CREWS framework [10], but there they only make a distinction between instance and type scenario. It is important to notice that our use of scenarios is of the typed scenario as per the cited framework; that is, we treat actors and events by their collective name and not by particular instances.

It is important once more to stress that in our view scenarios describe situations in the macrosystem and not only a sequence of interactions between a user and a system. Prior work on scenario integration [49] used a user-interaction point of view and focused on the behavioural aspect of scenarios. Glinz used Statecharts as a means to model scenarios, exploring not only the capability of decomposition of Statecharts but also the concurrence aspects, thus making possible not only the integration of several scenarios in one, but also the analysis of the scenarios for consistency problems. Contrary to Glinz [49], our approach to scenario integration deals with the behavioural aspect of scenarios (episodes), but with other context information as well, such as actors, resources, goals and preconditions. On the other hand, our proposal is strongly based on heuristics that, as of now, does not have automated support.

Desharnais et al. [51] describe a state-oriented relational approach to scenario and applies it to sequential scenarios. Scenarios here are atomic descriptions, in a Z style, of the interaction between the system and the environment represented by a triple (T, Re, Rs), where T is a space and Re and Rs are disjoint relations on T. The approach assumes that the scenarios fulfil certain consistency descriptions. From the atomic scenarios, a functional specification is derived. This approach seems promising, once you get to the level of these atomic descriptions, which in our understanding are of a level even lower than our episodes.

One of the first articles dealing with scenario organisation came from the object-oriented community [35]. In this article Cockburn presents the problems of 'scenario explosion' and proposes a hierarchy of goals as a way of solving it. Regnell [52] also deals with the problem in a goal-oriented way; his proposal organises the use cases in a top-down fashion (environment level, structure level, event level), but he also points out that to better understand a user case there should be a synthesis phase were use cases are integrated in abstract usage scenarios.

The CREWS–L'Ecritoire [53, 54] approach uses a combination of goals and scenarios not only to elicit but also to organise scenarios. From goal elicitation, by case-based discovery, a template-driven strategy with free prose leads to scenarios. Once scenarios are available they are organised in a hierarchical way, also based on goal modelling (AND/OR relationships). They also provide some heuristics for quality management [54]. Comparing this with our work, we may say that we directly use situations, in a bottom-up fashion, to understand the problem and when organising the scenarios we may integrate them with our integration mechanisms that encapsulate the notions of AND/OR relationships.

Sutcliffe et al. [55] present a method for scenario-based RE. Its basic representation is similar to ours, with the difference that we have scenario sub-scenarios hierarchies (*n* levels of structuring) and they have just one level of structuring; that is, each use-case can have *m* scenarios. They also present a method as a DFD diagram [55, p. 1074, Fig. 1], where four steps are presented: elicit use case, analyse generic problems, generate scenarios and validate scenarios. We have shown here a much more detailed process, focusing on elicitation of scenarios from the connotation of the application vocabulary.

We would like to stress that our results are based on real experience with the use of scenarios. We firmly believe that the presentation of our strategy for building scenarios is important to researchers as well as for practitioners, since the information we provided helps the general understanding of how scenarios are produced and how they might get organised. We will continue studying the abstraction facet of scenarios and, in particular, we are looking at the other axis of abstraction mechanisms, namely specialisation/generalisation, in order to explore how this type of organisation could support scenario reusability.

Acknowledgments. We would like to thank the reviewers for their time and careful reading of our first submission. We hope that by addressing their comments the paper is more readable. We would like to thank Karin Koogan Breitman for her comments. Financial support for this work was partially provided by Universidad de Belgrano, Conicet, Faperj and CNPq.

References

1. Leite JCSP, Freeman PA. Requirements validation through viewpoint resolution. IEEE Trans Software Eng 1991; 17(12):1253–1269
2. Leite JCSP, Rossi G, Balaguer F, Maiorana V, Kaplan G, Hadad G, Oliveros A. Enhancing a requirements baseline with scenarios. Requirements Eng 1997;l2(4):184–198
3. Jackson M. Software requirements and specifications: a lexicon of practice, principles and prejudices. Addison-Wesley, Reading, MA/ACM Press, New York, 1995
4. Loucopoulos P, Karakostas V. System requirements engineering. McGraw-Hill, London, 1995
5. Carroll J. Introduction: the scenario perspective on system development. In: Carroll J (ed). Scenario-based design: envisioning work and technology in system development. Wiley, New York, 1995, pp 1–18
6. Potts C. Using schematic scenarios to understand user needs. In: Proceedings of DIS'95 – Symposium on designing interactive systems: processes, practices and techniques. ACM Press/ University of Michigan, 1995, pp 247–256
7. Booch G. Object oriented design with applications. Benjamin Cummings, Redwood City, CA, 1991
8. Jacobson, I, Christerson, M, Jonsson, P, Overgaard, G. Object-oriented software engineering: a use case driven approach. Addison-Wesley, Reading, MA/ACM Press, New York, 1992
9. Zorman L. Requirements envisaging by utilizing scenarios (Rebus). PhD dissertation, University of Southern California, 1995
10. Rolland C, Ben Achour C, Cauvet C, Ralyté J, Sutcliffe A, Maiden M, Jarke M, Haumer P, Pohl K, Dubois E, Heymans P. A proposal for a scenario classification framework. Requirements Eng 1998;3(1):23–47
11. Weidenhaupt K, Pohl K, Jarke M, Haumer P. Scenarios in system development: current practice. IEEE Software 1998;34–45
12. Rolland C, Ben Achour C. Guiding the construction of textual use case specifications. Data Knowl Eng 1998;25:125–160
13. Sutcliffe A. Workshop: exploring scenarios in requirements engineering. In: Proceedings of the third IEEE international symposium on requirements engineering. IEEE Computer Society Press, Los Alamitos, CA, 1997, pp 180–182
14. Breitman KK, Leite JCSP. A framework for scenario evolution. In: Proceedings of the IEEE international conference on requirements engineering. IEEE Computer Society Press, Los Alamitos, CA, 1998, pp 214–221
15. Potts C, Takahashi K, Antón AI. Inquiry-based requirements analysis. IEEE Software 1994;11(2):21–32
16. Booch G. Scenarios. Rep Object Analysis Design 1994; 1(3):3–6
17. Firesmith DG. Modeling the dynamic behavior of systems, mechanisms, and classes with scenarios. Rep Object Analysis Design 1994;1(2):32–36
18. Sutcliffe A. A technique combination approach to requirements engineering. In: Proceedings of the third IEEE international symposium on requirements engineering. IEEE Computer Society Press, Los Alamitos, CA, 1997, pp 65–74
19. Hsia P, Samuel J, Gao J, Kung D, Toyosima I, Chen C. Formal approach to scenario analysis. IEEE Software 1994;7(2):33–41
20. Dano B, Briand H, Barbier F. An approach based on the concept of use case to produce dynamic object-oriented specification. In: RE'95: Proceedings of the third IEEE international symposium on requirements engineering. IEEE Computer Society Press, Los Alamitos, CA, 1997, pp 54–64
21. Robertson SP. Generating object-oriented design representations via scenario queries. In: Carroll J (ed). Scenario-based design: envisioning work and technology in system development. Wiley, New York, 1995, pp 279–306

22. Wirfs-Brock R. Designing objects and their interactions: a brief look at responsibility-driven design. In: Carroll J (ed). Scenario-based design: envisioning work and technology in system development. Wiley,, New York, 1995, pp 337–359
23. Oberg R, Probasco L, Ericsson M. Applying requirements management with use cases. Rational Software Corporation, 1998
24. Schneider G, Winters J. Applying use cases: a practical guide. Addison-Wesley, Reading, MA, 1998
25. Constantine L. Joint essential modeling, user requirements modeling for usability. In: International conference on requirements engineering (Tutorial Notes). Constantine & Lockwood, Colorado Springs, CO, 1998
26. Jacobson I. Basic use-case modeling. Rep Object Analysis Design 1994;1(2):15–19
27. Jacobson I. Basic use-case modeling (continued). Rep Object Analysis Design 1994;1(3):7–9
28. Gough PA, Fodemski FT, Higgins SA, Ray SJ. Scenarios: an industrial case study and hypermedia enhancements. In: RE'95: Proceedings of the international symposium on requirements engineering. IEEE Computer Society Press, Los Alamitos, CA, 1995, pp 10–17
29. Leite JCSP, Franco APM. O Uso de Hipertexto na Elicitaçao de Linguagens da Aplicaçao. In: Anais de IV Simpósio Brasilero de Engenharia de Software. SBC, Brazil, 1990, pp 134–149
30. Hadad G, Kaplan G, Oliveros A, Leite JCSP. Integración de Escenarios con el Léxico Extendido del Lenguaje en la elicitación de requerimientos: Aplicación a un caso real. Rev Inform Teórica Aplicada (RITA), Brazil 2000;6(1):77–104
31. van Lamsweerde A, Darimont R, Massonet Ph. The meeting scheduler system: preliminary definition. Internal report, Université Catholique de Louvain, 1993
32. Leite JCSP. Eliciting requirements using a natural language based approach: the case of the meeting scheduler. In: Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1993
33. Leite JCSP, Franco APM. A strategy for conceptual model acquisition. In: IEEE international symposium on requirements engineering. IEEE Computer Society Press, Los Alamitos, CA, 1993, pp 243–246
34. Chen P. The entity-relationship model: towards a unified view of data. ACM Trans Database Syst 1976;1(1):9–36
35. Cockburn A. Structuring use cases with goals. Technical report, Humans and Technology, Salt Lake City, UT (http://members.aol.com/acockburn/papers/usecases.htm), 1995
36. Breitman KK, Leite JCSP. Processo de Software baseado en Cenários. In: Proceedings of WER'99, Workshop en Requerimientos, Buenos Aires, Argentine, 1999, pp 95–105
37. Parnas DL, Clements PC. A rational design process: how and why to fake it. IEEE Trans Software Eng 1986;SE-12(2):251–257
38. Ross D, Schoman A. Structured analysis for requirements definition. IEEE Trans Software Eng (special issue on requirements analysis) 1977;3(1):6–15
39. Goguen JA, Linde Ch. Techniques for requirements elicitation. In: Proceedings of the international symposium on requirements engineering. IEEE Computer Society Press, Los Alamitos, CA, 1992, pp 152–164
40. Doorn J, Kaplan G, Hadad G, Leite JCSP. Inspección de Escenarios. In: Proceedings of WER'98, Workshop en Engenharia do Requisitos, Maringá, Brazil, 1998, pp 57–69
41. Hadad G, Kaplan G, Oliveros A, Leite JCSP. Construcción de Escenarios a partir del Léxico Extendido del Lenguaje, Buenos Aires, Argentine, 1997, pp 65–77
42. Leite JCSP, Oliveros A, Rossi G, Balaguer F, Hadad G, Kaplan G, Maiorana V. Léxico extendido del lenguaje y escenarios del sistema nacional para la obtención de pasaportes. Technical report 7, Departamento de Investigación, Universidad de Belgrano, Buenos Aires, 1996
43. Hadad G, Kaplan G, Oliveros A, Leite JCSP. Léxico Extendido del Lenguaje y Escenarios del Meeting Scheduler. Technical report 13, Departamento de Investigación, Universidad de Belgrano, Buenos Aires, 1998
44. Rivero L, Doorn J, Del Fresno M, Mauco V, Ridao M, Leonardi C. Una Estrategia de Análisis Orientada a Objetos basada en Escenarios: Aplicación en un Caso Real. In: Proceedings of WER'98: Workshop en Engenharia do Requisitos, Maringá, Brazil, 1998, pp 79–90
45. Mauco V, Ridao M, del Fresno M, Rivero L, Doorn J. Ingeniería de Requisitos, Proyecto: Sistema de Planes de Ahorro. Technical report, ISISTAN, UNCPBA, Tandil, Argentine, 1997
46. García O, Gentile CG. Diseño de una herramienta para contrucción de LEL y Escenarios. Graduation dissertation, Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentine, 1999
47. Rubin KS, Goldberg J. Object behavior analysis. Commun ACM 1992; 35(9):48–62
48. Dardenne A, van Lamsweerde A, Fickas S. Goal directed requirements acquisition. Sci Comput Prog 1993;20:3–50
49. Glinz M. An integrated formal model of scenarios based on statecharts. In: Proceedings of the 5th ESEC, Lecture Notes on Computer Science 989. Springer, Berlin, 1995, pp 254–271
50. Jarke M, Bui TX, Carroll JM. Scenario management: an interdisciplinary approach. Requirements Eng 1998;3(4):155–173
51. Desharnais J, Frappier M, Khedri R, Mili A. Integration of sequential scenarios. IEEE Trans Software Eng 1998;24(9):695–708
52. Regnell B. Hierarchical use case modeling for requirements engineering. Department of Communication Systems, Lund Institute of Technology, Lund University, Sweden, 1996
53. Rolland C, Souveyet C, Ben Achour C. Guiding goal modeling using scenarios. IEEE Trans Software Eng 1998;24(12):1055–1071
54. Ben Achour C, Rolland C, Souveyet C. A proposal for improving the quality of scenario collections. In: Proceedings of the 4th international workshop on RE: foundations of software quality, REFSQ'98. Presses University of Namur (http://sunsite.informatik.rwth-aachen.de/CREWS), 1998, pp 29–42
55. Sutcliffe AG, Maiden NAM, Minocha S, Manuel D. Supporting scenario-based requirements engineering. IEEE Trans Software Eng 1998;24(12):1072–1088