A Requirements Elicitation Approach Based in Templates and Patterns*

A. Durán Toro, B. Bernárdez Jiménez, A. Ruiz Cortés, and M. Toro Bonilla

Departamento de Lenguajes y Sistemas Informáticos, Facultad de Informática y Estadística,
Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, España
{amador,beat,aruiz,mtoro}@lsi.us.es

Abstract One of the main problems of requirements elicitation is expressing customer requirements in a form that can be understood not only by requirements engineers but also by noncomputer professional customers and users. The usual choice for expressing elicited requirements is natural language, since it is frequently the only common language to all participants. Problems of natural language are well–known, but using more formal notations too early is a risky choice that can make requirements impossible to understand for customers and users. Moreover, requirements engineers do not usually have good writing skills, and sometimes *semantically correct* requirements, expressed in natural language, are not understood because of the way they are written. In this paper, we present requirements templates that can improve requirements elicitation and expression, and two kinds of patterns: *linguistic patterns*, which are very used sentences in natural language requirements descriptions that can be parameterized and integrated into templates, and *requirements patterns*, which are generic requirements templates that are found very often during the requirements elicitation process and that can be reused with some adaptation.

Keywords: requirements engineering, requirements elicitation

1 Introduction

Following [15], requirements elicitation can be defined as the process through which customers and users of a software system discover, reveal, articulate, and understand their requirements. According to [3], problems of requirements elicitation can be classified into three main groups: problems of *scope*, i.e. deciding the boundary of the system and avoiding unnecessary information, problems of *understanding* between the communities involved in the process, and problems of *volatility* since requirements evolve over time.

One of the main problems of understanding is expressing and recording the requirements in a form that can be understood not only by requirements engineers but also by noncomputer professional customers and users [5, Principle 56]. Current elicitation techniques such as Joint Application Development (JAD), brainstorming or interviews

* This work is funded by the CICYT project "MENHIR". TIC 97-0593-C05-01, Ministry of Education and Science (Spain)

[15] do not address requirements expression. For example, in JAD and brainstorming sessions, elicited requirements are supposed to remain visible to the participants, but the way those requirements are expressed is not described by these elicitation techniques.

The usual choice for expressing elicited requirements is natural language, since it is frequently the only common language to customers, users and requirements engineers. Problems of natural language are well–known, but using more formal notations too early is a risky choice that can make requirements impossible to understand for customers and users [5, Principles 54, 55 and 56]. Since requirements engineers do not usually have good writing skills, sometimes requirements expressed in natural language are not understood because of the way they are written [5, Principle 51].

In this paper, we present requirements templates and patterns that can help requirements engineers and users to elicit, express and record information systems requirements using natural language. We have developed requirements templates and identified two kinds of patterns: *linguistic patterns* (*L–patterns*), which are very used sentences in natural language requirements descriptions that can be parameterized and integrated into templates, and *requirements patterns* (*R–patterns*), which are generic requirements templates that are found very often during the requirements elicitation process and that can be reused with some adaptation.

The structure of this paper is as follows. In section 2 we present the requirements engineering model that will be followed in the rest of the paper. In section 3 we present templates and patterns for the three kinds of information systems requirement we have identified. In section 4, a prototype of a CASE tool supporting requirements templates and an object–oriented model of requirements are briefly presented. Finally, in section 5 some related works are compared, and in section 6 some conclusions are given and some future work is pointed out.

2 Requirements Engineering Model

In this paper, we will follow the requirements engineering model, and its associated terminology, shown in Fig. 1. This model is partially based in the one proposed in [15]. The meaning of each process is the following:

- **Requirements Elicitation**: requirements are elicited from customers and users using elicitation techniques such as interviews, JAD or brainstorming [3] and other auxiliary techniques such as *in situ* research, document analysis, forms or the templates and patterns proposed in this paper. The results of this process are the system requirements also known as user requirements or *Customer–Oriented Requirements*, shortly, *C–requirements* [1].
- **Requirements Analysis**: C–requirements are analyzed in order to detect inconsistencies and identify missing requirements, usually by building an object–oriented o structured model. In this process, in which customers and users can participate provided they have been trained in modeling techniques, C–requirements are transformed into software requirements, also known as *Developer–Oriented Requirements* or, shortly, *D–requirements* [1]. Other usual product of this process is a prototype of the system to be built, which, in case executable formal specification were used to express D-requirements, might be automatically generated.

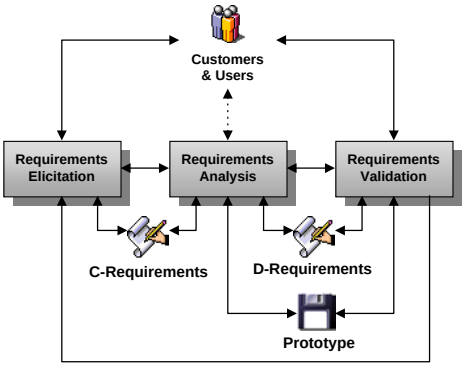

Figure1. Requirements Engineering Model

– **Requirements Validation**: customers and users must validate the requirements and evaluate the prototype, usually leading to elicitation of new requirements. The whole process iterates until all requirements are validated and no more requirements are elicited.

3 Requirements Templates and Patterns

Requirements templates help requirements to be expressed. Requirements information is structured in a fixed form, so requirements engineers know what missing information must be searched, requirements can easily be treated by a software tool (see Fig. 9 in section 4) and reuse is promoted. In addition, filling blanks in pre–written sentences, i.e. L–patterns, is easier and faster than writing a whole paragraph saying what the system is expected to do. Moreover, whole requirements templates, i.e. R–patterns, can be reused many times, provided they have been identified, with some adaptation to specific developments.

In next sections, requirements templates and patterns for information systems are described. The used notation is the following: words between < and > must be properly replaced, words between { and } and separated by commas represents options; only one option must be chosen.

3.1 Information Storage Requirements

Information Storage Requirements Template and L–patterns The most important thing in information systems is information. The template for information storage requirements, see Fig. 2, helps users to answer the question "*what information, relevant for your business goals, must be stored by the system*?". The meaning of the template fields is the following:

– **Identifier and descriptive name**: every requirement must be uniquely identified by a number and a descriptive name [5, Principle 52]. In order to help rapid identification, information storage requirements identifiers start with *RI*.

RI–<*id*>	<***descriptive name***>
Version	<*current version number*> (<*current version date*>)
Author	<*current version author*> (<*author's organization*>)
Source	<*current version source*> (<*source's organization*>)
Purpose	<*purpose of requirement*>
Description	The system shall store the information corresponding to <*relevant concept*>. More precisely:
Specific data	• <*specific data about the relevant concept*> • . . .
Time interval	{ past and present, only present }
Importance	<*importance of requirement*>
Urgency	<*urgency of requirement*>
Comments	<*additional comments about the requirement*>

Figure2. Template and L–patterns for Information Storage Requirements

- **Version**: following IEEE recommendations [12], different versions of requirements must be managed. This field contains the current version number and date of the requirement.
- **Author, Source**: these fields must contain the name and organization of the author, i.e. the requirements engineer, and the source, i.e. the user or customer, of the current version of the requirement.
- **Purpose**: this field must state why the requirement is necessary to achieve business goals. [5, Principle 43].
- **Description**: for information storage requirements, this field uses an L–pattern that must be completed with the *relevant concept* about information must be stored.
- **Specific data**: this field must hold a list of specific data associated with the relevant concept.
- **Time interval**: this field indicates how long information about the concept is relevant for the system. It can takes two values: *past and present*, if information is always relevant, and *present only* if information has a valid period of time. For example, if the concept is *employees*, a *past and present* time interval means that ex–employees are relevant for the system; a *present only* time interval means that ex–employees are not under consideration.
- **Importance, Urgency**: these fields indicate how important and urgent the requirement is for customers and users [5, Principle 50]. They can be assigned a numeric value or some enumerated expressions such as *vital*, *important* or *would be nice* for importance, or such as *immediately*, *under pressure* or *can wait* for urgency, as proposed in [14].
- **Comments**: other information about the requirement that cannot be fitted in previous fields can be recorded here.

An example of use of this template, supposing a video tape renting system, is shown in Fig. 3.

RI–01	**Information about movies**
Version	1.0 (Feb, 17, 1999)
Author	A. Durán (University of Seville)
Source	R. Corchuelo (Super Video Shop)
Purpose	To know availability of movies at any moment and to be able to help customers to select a movie using different criteria
Description	The system shall store the information corresponding to movies in the video store. More precisely:
Specific data	• Title of the movie • Number of tapes of the movie rented at any moment • Number of tapes of the movie ready to rent at any moment • Type of the movie: children, action, science–fiction or adults • Time of the movie, in hours and minutes • Main actors of the movie • Director of the movie • Producer of the movie • Year of production of the movie
Time interval	past and present
Importance	vital
Urgency	immediately
Comments	none

Figure3. Example of Information Storage Requirement

Information Storage Requirements R–patterns After using the templates and patterns described in this paper in more than 40 academic practices in the field of information systems, we have realized that there are very similar requirements that are present in most developments. For information storage requirements, we have identified some R–patterns such as those referring to information about customers (see Fig. 4), products, orders, invoices, etc. These R–patterns can be classified according to different criteria and stored in a repository for further reuse.

RI–x	**Information about *customers***
...	...
Description	The system shall store the information corresponding to *customers*. More precisely:
Specific data	• Legal identification number of *customer* • Name of *customer* • Address of *customer* • Telephone numbers of *customer* • E–mail address of *customer*
...	...

Figure4. Example of Information Storage R–pattern

RF–<*id*>	<***descriptive name***>		
Version	<*current version number*> (<*current version date*>)		
Author	<*current version author*> (<*author's organization*>)		
Source	<*current version source*> (<*source's organization*>)		
Purpose	<*purpose of requirement*>		
Description	The system shall behave as described in the following sequence of interactions when <*triggering event*>		
Precondition	<*precondition of use case*>		
Ordinary sequence	**Step**	**Action**	
	…	…	
	n	{The {<*actor*>, system} <*action performed by actor/system*>, Steps described in <*use case (RF–x)*> are performed}	
		$n.1$	If <*condition*>, {the {<*actor*>, system} <*action performed by actor/system*>, steps described in <*use case (RF–x)*> are performed}
		…	…
	…	…	
Postcondition	<*postcondition of use case*>		
Exceptions	**Step**	**Action**	
	p	If <*exception condition*>, {the {<*actor*>, system} <*action performed by actor/system*>, steps described in <*use case (RF–x)*> are performed}, then the sequence is {resumed, aborted}	
	…	…	
Performance	**Step**	**Maximum time**	
	q	*m* seconds	
	…	…	
Frequency	This use case is expected to be performed <*number of times*> times/<*time unit*>		
Importance	<*importance of requirement*>		
Urgency	<*urgency of requirement*>		
Comments	<*additional comments about the requirement*>		

Figure5. Template and L–patterns for Functional Requirements (Use Cases)

3.2 Functional Requirements

Functional Requirements Template and L–patterns Information systems not only store information, they must also provide services using the information they store. The functional requirements template, see Fig. 5, describes use cases [13], and help users and customers to answer the question "*what do you want the system to do with the stored information in order to achieve your business goals?*". The meaning of the template fields is the following:

- **Identifier and descriptive name**: the same as in information requirements template, except that functional requirements identifiers start with *RF*.
- **Version, Author, Source, Purpose**: the same as in information storage requirements.

- **Description**: for functional requirements, this field contains an L–pattern that must be filled with the *triggering event* that starts the use case.
- **Precondition**: necessary conditions that must hold in order to perform the use case are expressed here.
- **Ordinary sequence**: this field holds the ordinary sequence of interactions of the use case. In every step, one actor or the system can perform an action, or other use case can be performed, i.e. *used*, following the semantics of *uses* and *extends* relationships given in [17]. A step can have conditional substeps, assuming that only one substep is performed. Other use cases can be performed in conditional substeps, i.e. the use case can be *extended*.
- **Postcondition**: conditions that must hold after normal termination of the use case are expressed here.
- **Exceptions**: after performing a step of the use case, some exceptional conditions may arise. This field of the template specifies the behavior of the systems in such circumstances. After the action or the use case associated with the exception (i.e. the *extender* use case) is performed, the system can resume the ordinary sequence or aborts the use case.
- **Performance**: for any step or substep in which an action is performed by the system, a maximum time can be specified in this field.
- **Frequency**: although frequency is not actually a requirement, it is an important information for developers and can be recorded here.
- **Importance, Urgency and Comments**: the same as for information requirements template.

An example of use of this template, supposing the same previous video tape renting system, is shown in Fig. 6.

Functional Requirements R–patterns For functional requirements, we have identified four R–patterns which are always present in every information system development and that we have named *CRUD* R–patterns (***C**reate, **R**ead, **U**pdate, **D**elete*). These R–patterns must always be present in correct information systems: information stored in the system must be *created* (see Fig. 7) and *updated* in order to be synchronized with its environment; obsolete information must be *deleted* if we do not want to run out of storage space; and finally, some people must be able to *read* the stored information and use it.

3.3 Non–Functional Requirements

Non–functional Requirements Template and L–patterns Other capabilities of the system, such as privacy, reliability, etc. can be expressed using the non–functional requirements template. An example can be seen in Fig. 8. This template does not have any specific field, since it is a generic template. The only identified L–pattern, for the moment, is used in the description field and its form is: *The system shall <system capability>*.

RF–07	**Customer returns video tape(s)**	
Version	2.1 (Feb, 10, 1999)	
Author	B. Bernárdez (University of Seville)	
Source	A. Ruiz (Super Video Shop)	
Purpose	To control tape returns and customers payments	
Description	The system shall behave as described in the following sequence of interactions when a customer wants to return one or more video tapes	
Precondition	all video tapes are Super Video Shop tapes	
Ordinary sequence	**Step**	**Action**
	1	The clerk requests the system to start the return procedure
	2	The system requests for tape(s) identification(s)
	3	The clerk provides all identifications needed
	4	The system calculates the amount and prints the invoice
	4.1	If any tape is lately returned, the system charges an extra of 10% for every late return
	5	The customer pays the invoice
	6	The clerk puts the tape(s) on the shelves
Postcondition	stored information is updated, tapes are ready to rent again	
Exceptions	**Step**	**Action**
	3	If some tape is not registered as rented, the system reports the situation to the clerk and does not include the tape in the invoice, then the sequence is resumed
Performance	**Step**	**Maximum time**
	4	5 seconds
Frequency	This use case is expected to be performed 50 times/day	
Importance	vital	
Urgency	immediately	
Comments	none	

Figure6. Example of Functional Requirement (Use Case)

RF–x	**{Create, Register}** <***new information***>	
...	...	
Precondition	<*new information*> is not stored yet	
Ordinary sequence	**Step**	**Action**
	1	The <*some actor*> requests the system to start the <*create new information*> procedure
	2	The system requests for <*new information*>
	3	The <*some actor*> provides <*new information*>
Postcondition	<*new information*> is stored	
...	...	

Figure7. Example of Functional Requirement R–pattern (*Create*

RN–3	**Operating System**
Version	1.0 (Jan, 15, 1999)
Author	A. Durán (University of Seville)
Source	M. Toro (Super Video Shop)
Description	The system shall operate under the Linux Operating System
Importance	vital
Urgency	immediately
Comments	Check different Linux versions compatibility

Figure8. Example of Non–Functional Requirement

Non–functional Requirements R–patterns Having integrated performance requirements into the template described in section 3.2, not many R–patterns for non–functional requirements have been identified. The example of Fig. 8 can be used as a R–pattern for specifying the operating system under the system must be able to operate if its description field is changed into: *The system shall operate under <operating system>*.

4 CASE Tool Support

Currently, a CASE tool based on the object–oriented model of user requirements presented in [10] and shown in Figs. 10, 11 and 12 is under development. The CASE tool prototype is a document—based application, considering a requirements project as a document composed by a Customer Requirements Document and a Developer Requirements Document. The tool presents different views of the requirements project, as can be seen in Fig. 9. The user can add objects in every view and see the final documentation in a WYSIWYG–like fashion using HTML (the documentation window is actually a web browser). Generating documentation using HTML make possible to publish electronically in the web, so geographically distant participants can always have up–to–date information. Requirements project objects are stored in a relational database, using the techniques described in [7], so it is possible to access them from other applications.

5 Related Work

The idea of using templates for expressing requirements is based in the use case templates by Rumbaugh [17] and, mainly, by Cockburn [4] and his use case template. We have extended Cockburn's ideas to other types of requirement, not only functional requirements, have integrated some L–patterns into the templates and have identified several R–patterns. Inspired by [11], requirements patterns have been a *natural product* of our experience in requirements engineering.

Other similar works are the *Volere Requirements Specification Template* [16], that defines many types of non–functional requirements, and the *User Requirements Document Template* for the European Space Agency (ESA) PSS–05 standard developed at CERN [2].

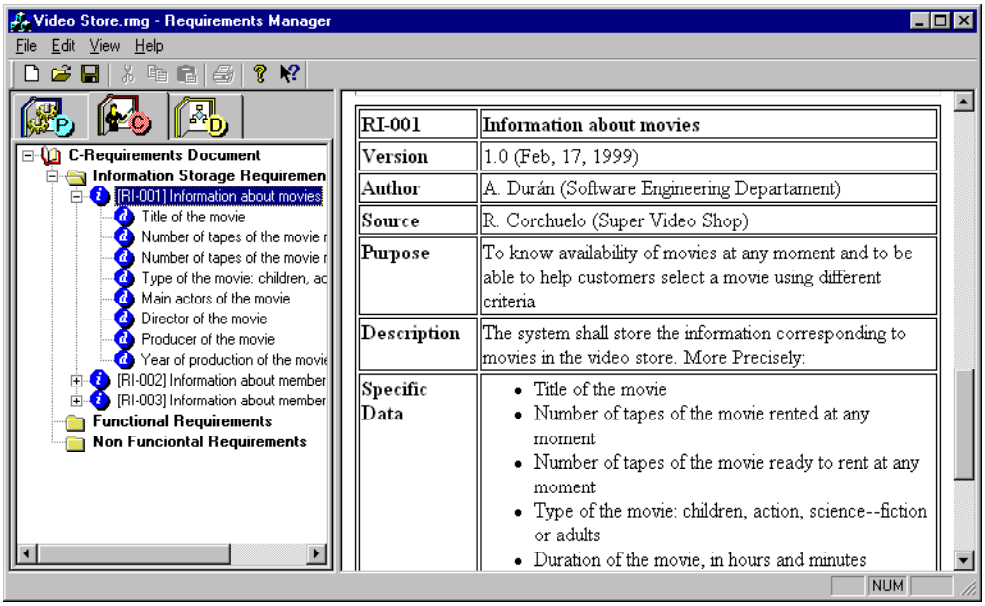

Figure9. Customer Requirements Document View of the CASE Tool Prototype

Ideas presented here have been integrated into an requirements elicitation methodology presented in [9] and have been adapted for the Spanish Government's structured methodology MÉTRICA [6]. These ideas have also been recently presented in [8].

6 Conclusions and Future Work

In this paper, we have presented requirements templates and patterns for information systems that can help to elicit and express requirements while keeping the benefits from

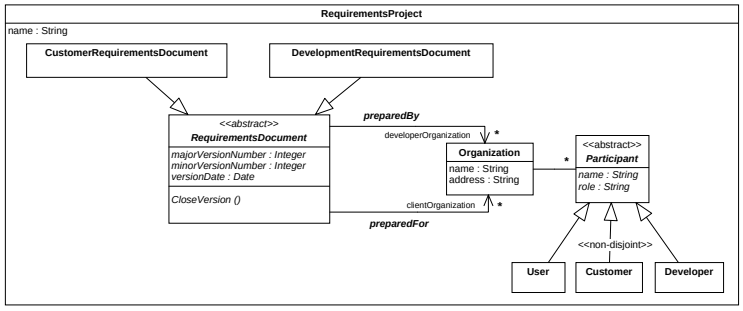

Figure10. Requirements Engineering Project Model

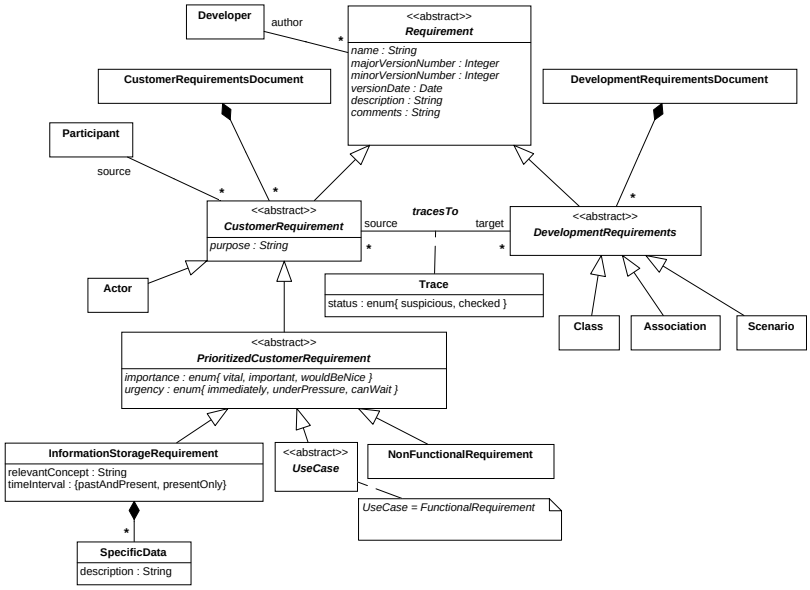

Figure11. C–Requirements Model

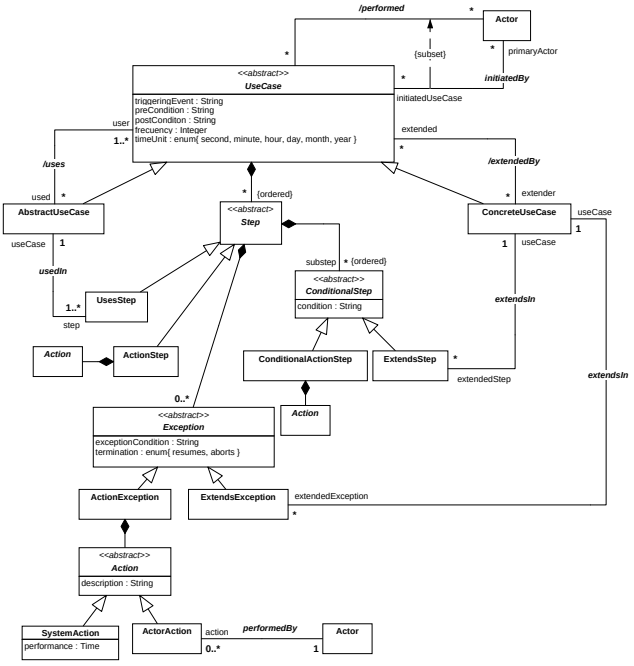

Figure12. Use Cases (Functional Requirements) Model

using natural language and avoiding early formalization of requirements. As a result, customers and users *do* understand requirements.

These templates and patterns have been successfully applied in more than 40 academic practices and are currently being successfully used in two real developments of Sadiel S.A., a top software company of Andalucía (Spain), where their use have dramatically improved communication with customers and users, changing the focus to requirements semantics, instead of how semantics are expressed. Sadiel uses the Spanish Government's structured methodology MÉTRICA, very similar to SSADM, and the templates and patterns presented here have been successfully integrated into this methodology, following our proposal in [6].

Some possible lines for future work can include adapting templates when more feedback from real developments is available, discovering more patterns, specially for non–functional requirements, creating a requirements repository for promoting reuse, finishing the CASE tool and generating documentation in XML format.

References

[1] J. W. Brackett. Software Requirements. Curriculum Module SEI–CM–19–1.2, Software Engineering Institute, Carnegie Mellon University, 1990.

[2] CERN. PSS–05 User Requirements Document Template. Technical report, CERN, 1998.

[3] M. G. Christel and K. C. Kang. Issues in Requirements Elicitation. Technical Report CMU/SEI–92–TR–12, Software Engineering Institute, Carnegie Mellon University, 1996.

[4] Alistair Cockburn. Goals and Use Cases. *Journal of Object–Oriented Programming*, Sep–Oct 1997.

[5] A. M. Davis. *201 Principles of Software Development*. McGraw–Hill, 1995.

[6] A. Durán, B. Bernárdez, M. Toro, and R. Corchuelo. A Proposal for the Requirements Catalog in MÉTRICA V2.1 (in Spanish). *Novática*, Accepted for publication 1999.

[7] A. Durán, A. Amaya, and M. Toro. Design of an Automatic Generator of Object–Relational Persistency Mechanisms. In *Actas de las II Jornadas de Ingeniería del Software*, San Sebastian, 1997.

[8] A. Durán, B. Bernárdez, M. Toro, R. Corchuelo, A. Ruiz, and J. Pérez. Expressing Customer Requirements Using Natural Language Requirements Templates and Patterns. In *IMACS/IEEE CSCC'99 Proceedings*, Athens, 1999. IMACS/IEEE.

[9] A. Durán, B. Bernárdez, M. Toro, and A. Ruiz. Una Propuesta Metodológica para la Elicitación de Requisitos de un Sistema Software. In *Actas de las III Jornadas de Trabajo Menhir*, Murcia, 1998. Universidad de Murcia.

[10] A. Durán, B. Bernárdez, M. Toro, and A. Ruiz. An Object–Oriented Model and a CASE Tool for Software Requirements Management and Documentation. In *Actas de las IV Jornadas de Trabajo Menhir*, Sedano (Burgos), 1999. Universidad de Valladolid.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley, 1995.

[12] IEEE. IEEE Guide for Developing System Requirements Specifications. IEEE/ANSI Standard 1233–1996, Institute of Electrical and Electronics Engineers, 1996.

[13] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object–Oriented Software Engineering: A Use Case Driven Approach*. Addison–Wesley, fourth edition, 1993.

[14] IBM OOTC. *Developing Object–Oriented Software: An Experience–Based Approach*. Prentice–Hall, 1996.

[15] S. Raghavan, G. Zelesnik, and G. Ford. Lecture Notes on Requirements Elicitation. Educational Materials CMU/SEI–94–EM–10, Software Engineering Institute, Carnegie Mellon University, 1994.

[16] James Robertson and Suzanne Robertson. Volere Requirements Specification Template Edition 6.0. Technical report, Atlantic Systems Guild, 1998.

[17] James Rumbaugh. Getting started: Using use cases to capture requirements. *Journal of Object–Oriented Programming*, September 1994.

Acknowledgments

We wish to thank Mr. César Pérez–Chirinos, from TransTools S.A., for his *CRUD* acronym and all his time spent discussing about patterns.