Requirements Eng (2009) 14:27–46
DOI 10.1007/s00766-008-0072-6

ORIGINAL ARTICLE

An integrated domain analysis approach for teleoperated systems

Joaquín Nicolás · Joaquín Lasheras · Ambrosio Toval · Francisco J. Ortiz · Bárbara Álvarez

Received: 6 August 2008 / Accepted: 3 December 2008 / Published online: 15 January 2009

Abstract Teleoperated systems for ship hull maintenance (TOS) are robotic systems for ship maintenance tasks, such as cleaning or painting a ship's hull. The product line paradigm has recently been applied to TOS, and a TOS reference architecture has thus been designed. However, TOS requirements specifications have not been developed in any rigorous way with reuse in mind. We therefore believe that an opportunity exists to increase the abstraction level at which stakeholders can reason about this product line. This paper reports an experience in which this TOS domain was analyzed, including the lessons learned in the construction and use of the TOS domain model. The experience is based on the application of extensions of well-known domain analysis techniques, together with the use of quality attribute templates traced to a feature model to deal with non-functional issues. A qualitative research method (action research) was used to carry out the experience.

Keywords Domain analysis · Product line requirements engineering · Feature modelling · Generic use cases · Teleoperated systems · Action research

J. Nicolás (✉) · J. Lasheras · A. Toval
Software Engineering Research Group,
Departamento de Informática y Sistemas,
Facultad de Informática, Universidad de Murcia,
Campus de Espinardo, 30071 Murcia, Spain
e-mail: jnr@um.es

F. J. Ortiz · B. Álvarez
Systems and Electronic Engineering Division,
Universidad Politécnica de Cartagena,
30202 Cartagena, Spain

1 Introduction

Teleoperated systems for ship hull maintenance (hereafter TOS) are robotic systems which are extremely useful in maintenance tasks such as cleaning or painting a ship's hull [1, 2]. Recent years have seen the development of a software reference architecture in the TOS domain [3]. Since TOS usually share a high number of common capabilities, this generic architecture provides a common framework for the reuse of software artefacts (*assets*). These systems can thus be considered to constitute a product line (or product family), i.e. they are a set of software-intensive systems which share a common, managed set of features that satisfy the specific needs of a particular market segment [4]. A comprehensive review of software product lines current practice is provided by van der Linden et al. [5] and Pohl et al. [6], while a recent vision of the challenges in the research in software product lines has been compiled by Käkölä and Dueñas [7].

Rine and Nada [8] have shown empirically that the level of reuse determines the effectiveness of the improvements in productivity, quality and time-to-market, and they conclude that greater benefits are obtained when reuse is considered during the early phases of the software development lifecycle. In TOS, in contrast, requirements specifications have not been developed rigorously with reuse in mind.

In this context, we propose the construction of new products in the TOS product line from a higher abstraction level—from the product line requirements instead of from its generic architecture—by developing what can be intuitively seen as a TOS *domain model* (the meaning of this and some related terms is discussed more rigorously in Sect. 4).

This paper presents an experience in analyzing the TOS domain, together with a set of lessons learned, and results from a TOS domain model that serves to (1) accelerate product line understanding (by indicating its common and variable elements), (2) obtain reusable requirements of the product line, (3) foster efficient decision-making in the specification of new systems, and (4) reuse the reference architecture and the commands used in the implementation of the product line. Well-known domain analysis techniques were chosen in order to improve the applicability of the proposal and the understanding of its results. These techniques have been set in a simple framework to represent the domain model, and have been extended to cope with the particularities of the domain. Domain analysis has been extended with quality attribute templates to specify non-functional concerns which are also traced to a feature model.

Glass et al. [9] point out that solutions for specific domains have traditionally received little attention in the software engineering literature, which has focused on generic solutions. This study, in contrast, presents an approach for modelling a specific domain, that of TOS. Glass et al. [9] also show that the research method most used in software engineering has been conceptual analysis, and they conclude that "software engineering researchers tend to analyse and implement new concepts, and they do very little or anything else". In contrast, this research uses action research [10], a proven research method (used primarily in the social sciences) in which the researchers work with the experts in the domain to generate a better understanding of a phenomenon.

The structure of this paper is as follows: Sect. 2 briefly describes the TOS domain and Sect. 3 introduces the research method used in the experience. Section 4 shows the framework used for domain analysis. This framework consists of a feature model (expanded in Sect. 5), a generic use case model (expanded in Sect. 6), a domain conceptual schema (expanded in Sect. 7), and quality attribute templates (expanded in Sect. 8). Section 9 offers the lessons learned and Sect. 10 reflects on the validity of the experience. Finally, Sect. 11 describes related work and Sect. 12 presents the conclusions and further work.

2 Teleoperated systems for ship hull maintenance

A critical operation for ship maintenance is periodical hull blasting before re-painting. To date some partial solutions exist, like blasting turbines for vertical surfaces or water blasting units for stripping. However, grit based solutions are usually restricted to full blasting in vertical surfaces, while the water based solutions are expensive and they have not shown such good performance and quality surface preparation as the grit blasting systems. In the European project EFTCoR (*environmental friendly and cost-effective technology for coating removal*, Fifth Framework Programme [11]), a family of robots (see Fig. 1) was developed for hull grit blasting which was capable of obtaining a high quality surface preparation together with a dramatic reduction of waste and zero emissions to the environment. Two research groups and eight European companies participated in this project.

Figure 2 shows a block diagram of the subsystems considered in the development of EFTCoR robots and their relationships. The diagram is a conceptual description in order to present the functionality of these systems briefly:

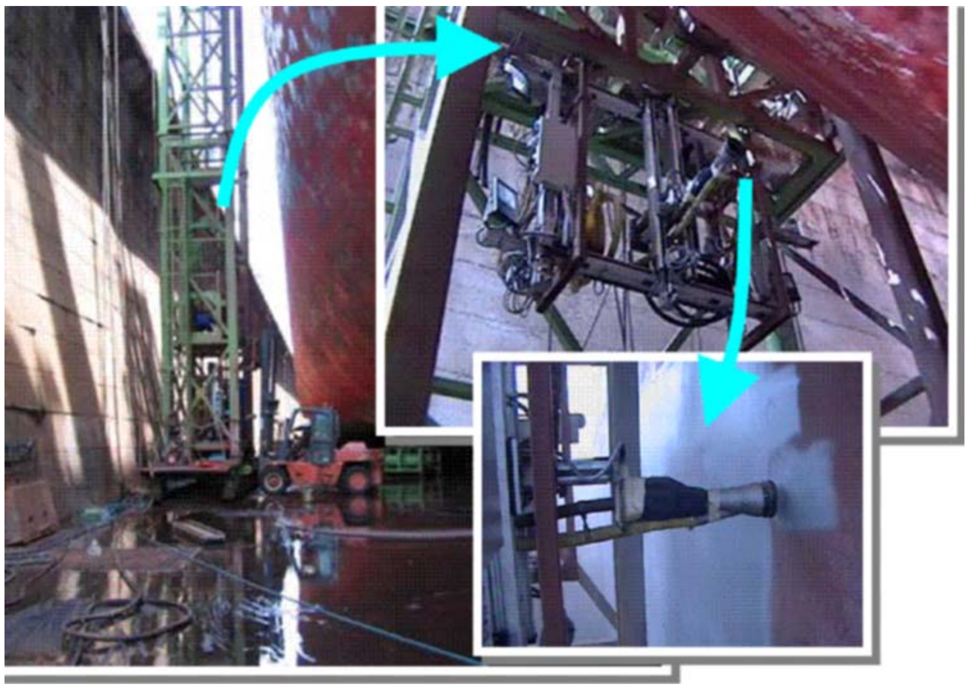


Fig. 1 A teleoperated system for ship hull coating removal

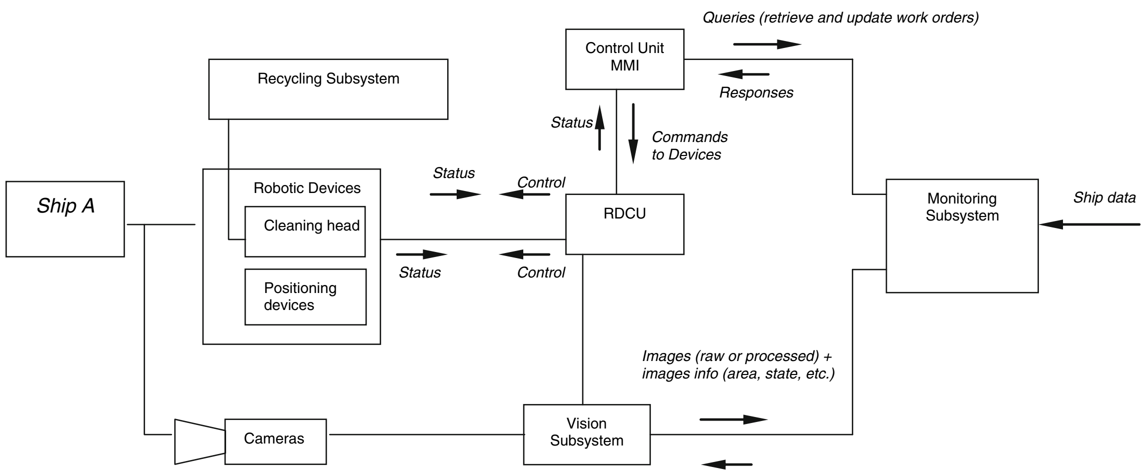


Fig. 2 Block diagram showing the TOS subsystems

- The *monitoring subsystem*, which encompasses the functionality concerning the informational and managerial needs related to ship maintenance.
- The *vision subsystem*, which comprises the functionality concerning hull inspection and determines the areas of the hull to be treated and their state before and after the surface preparation. This information constitutes one of the inputs to the *monitoring subsystem*.
- The *robotic device control unit (RDCU)*, which is in charge of controlling the robotic devices (positioning systems and tools) used in the maintenance tasks according to the orders introduced by an operator (by means of the *MMI control unit*) and the orders and events generated by other subsystems.
- The *recycling subsystem*, which is in charge of retrieving the residues from the working areas and recycling them. Because such residues have to be retrieved online, there is a strong relation between the operation of the blasting tool (*cleaning head*) and the operation of the Recycling Subsystem.

Robotic devices consist of a *cleaning head* and *positioning devices*. The positioning of the cleaning heads over the ship hull is a problem that can be solved in different ways. The following solution has been proposed by the EFTCoR project:

- A family of specialized, low cost systems has been developed instead of a single general purpose system.
- The different nature and requirements of the two cleaning operations, full blasting and spotting, lead to different systems.
- The global positioning system has been split into two independent positioning subsystems: a primary positioning system able to position heavy burdens along large surfaces (the whole ship hull) and a secondary positioning system, which can be mounted on the primary, able to position a light cleaning head with the precision required for spotting over small surfaces (4–10 m^2).
- The primary and secondary positioning systems, as well as their assemblies, have been designed in such a way that different combinations of primary and secondary positioning systems are possible.
- A climbing vehicle, *Lazaro*, provided with a cleaning head was developed to reach those areas that were unreachable with a reasonable combination of primary and secondary positioning systems. The vehicle can be used all over the ship hull.
- Whenever possible, a commercial solution has been adopted or adapted to do the work.

3 Research method

Action-research (A-R) is a qualitative approach which can be used to study the effects of changes in system development and maintenance methods. Baskerville [10] states that complex social processes (such as the use of information technologies in organizations) can be studied properly following an A-R approach, by introducing changes in those processes and observing the effects of those changes. A-R promotes a reflective learning process and a search for practical solutions that involves both researchers and practitioners. The application of A-R produces a cyclical process in which all the parties involved in the research participate by examining the existing situation with the intention of improving it. A-R does not refer to a concrete research method, but to a class of methods which, according to Baskerville [10], share the following features:

- Focus on a practical problem.
- Orientation towards action and change.

- Collaboration between participants.
- A process model that involves systematic and sometimes iterative steps.

In line with A-R terminology, the following roles have been used in this experience:

- The *researcher* is the Software Engineering Research Group of the University of Murcia (Spain).
- The *researched*, i.e. the object under research, is the TOS product line.
- The *critical reference group* (CRG), i.e. the group that has the problem we are trying to solve, is the Division of Electronics Engineering and Systems (DEES) of the Technical University of Cartagena (Spain). DEES performed technology transfer to naval companies like Navantia (formerly Izar) interested in the innovation that teleoperated systems provide to ship hull maintenance. DEES is directly in charge of the development of these robots.
- The *stakeholders* are all those organizations that might benefit from the results of the research: in this case, the CRG and, in general, companies that perform ship hull maintenance tasks, or manage similar teleoperated systems.

The present experience has been developed in the context of the 3-year R&D project described in Sect. 2. Reports of the work in progress were presented and discussed in five general 2-day workshops that took place during the project. In addition, there were four specific, 1-day meetings between the researcher and the CRG. Moreover, there were a lot of personal communications between the researcher and the CRG that were not formally registered. E-mail was used in most of these communications.

In this study, a *participative* application of A-R took place, in which the CRG put into practice the recommendations proposed by the researcher, with whom the effects and results were shared. Questionnaires were used to quantify the value for the CRG of the approach presented in this paper. An *empirical* variant of A-R would have been difficult to apply because it would have required the CRG to perform a wide systematic register of actions and effects, but records of the CRG's previous work did not exist, as is explained in Sect. 9.

The activities performed in the first cycle of this application of A-R are described in Fig. 3. The first cycle begins with a *plan*, in which the questions to guide the research are identified and the actions to solve those questions are specified. First, the interest of performing an analysis of the TOS domain was justified. Then the state of the art in the domain analysis field was studied in order to adapt the analysis models to tackle the problem. The study of the domain was shortened because the CRG had experience in

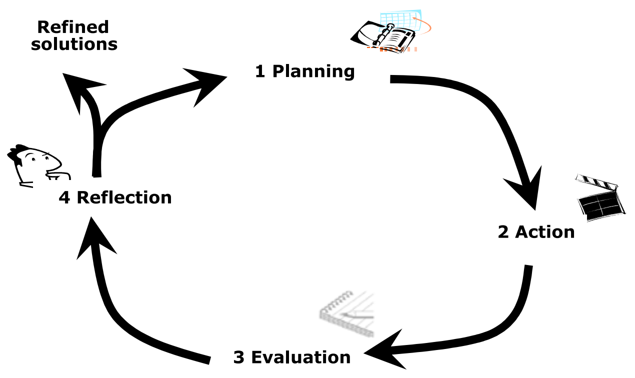


Fig. 3 A-R application process

developing TOS systems and they had a wide set of documents available describing the overall objectives and needs of these systems, their reference architecture, the specification of the commands used in their development, some informal safety and security concerns, and some early requirements documentation, including an initial conceptual model, an initial FODA feature model for a part of the product line and an attempt to use case identification. This documentation helped the researcher to gain knowledge of the domain.

After the plan, an *action* activity follows in which the researcher induces a careful, deliberate, and controlled variation of the practice. In this case study, the researcher and the CRG worked together closely to build a TOS domain model iteratively.

Next, an *observation* or *evaluation* is made, in which information on the effects of the action is collected, and some lessons learned begin to take shape. This observation was first performed in a purely qualitative manner, and then was supported by the analysis of a 20-point questionnaire that addressed the issues that had been considered of interest in the first qualitative assessment.

Finally, the first cycle ends with a *reflection*, in which the results are shared and analysed by all the stakeholders, and new interesting questions can be raised to be tackled in a second cycle, and so forth. This reflection was performed independently by the researcher and the CRG and a consensus was then reached.

4 A framework for domain analysis

First it was necessary to choose and characterize the models to use in TOS domain analysis. Olivé's work [12] describing the nature of conceptual schemas and the role that they play in information system development was used to unify terminology and provide a simple framework for this case study. Olivé focuses on information systems, but most of the conclusions may apply to the general field of software. According to this work, *conceptual schema* can be defined as the knowledge model that a system needs to perform its functions. The present experience tries to build a TOS conceptual schema.

Some confusion exists in the literature as to the similarities and differences between the notion of conceptual schema and similar concepts, such as *domain model*, *domain knowledge*, *functional specification* and *ontology*. Olivé's attempt is to knit different ideas to identify the nature of conceptual schemas and their roles in information systems development, and also to show the correspondence with related terms. Olivé considers that the conceptual schema of a system consists of two types of knowledge: (1) knowledge about its domain, and (2) knowledge about the functions that the system performs. The first is called *domain conceptual schema* (DCS), and the second *functionality specification* (FS). Figure 4 shows how these models have been devised in this case study.

On the one hand, the DCS, sometimes called *domain knowledge* or *domain model*, has been developed by means of:

- A UML class model, showing a taxonomy of domain entity types (with their attributes, relationships, and integrity constraints), and a set of domain event types. For instance, this model includes entities such as *tool*, *blasting*, *primary*, and *secondary*, and event types such as those that cause the system to change from one maintenance state to another (*apprenticeship*, *calibration*, *diagnosis*, and *configuration*). The conceptual model consists of about 80 conceptual classes.
- A UML state machine that describes the dynamics of the domain event types. There is only one state machine modelled to describe the allowed sequences of events changing the state of the system (eight states).

On the other hand, the FS has been carried out by means of:

- A feature model—cf. FODA, *feature-oriented domain analysis* [13], FORM, *feature-oriented reuse method* [14] or PLA, *product line analysis* [15]—, which intuitively specifies the vision of the product line that the stakeholders have. The whole model comprises about 150 features.

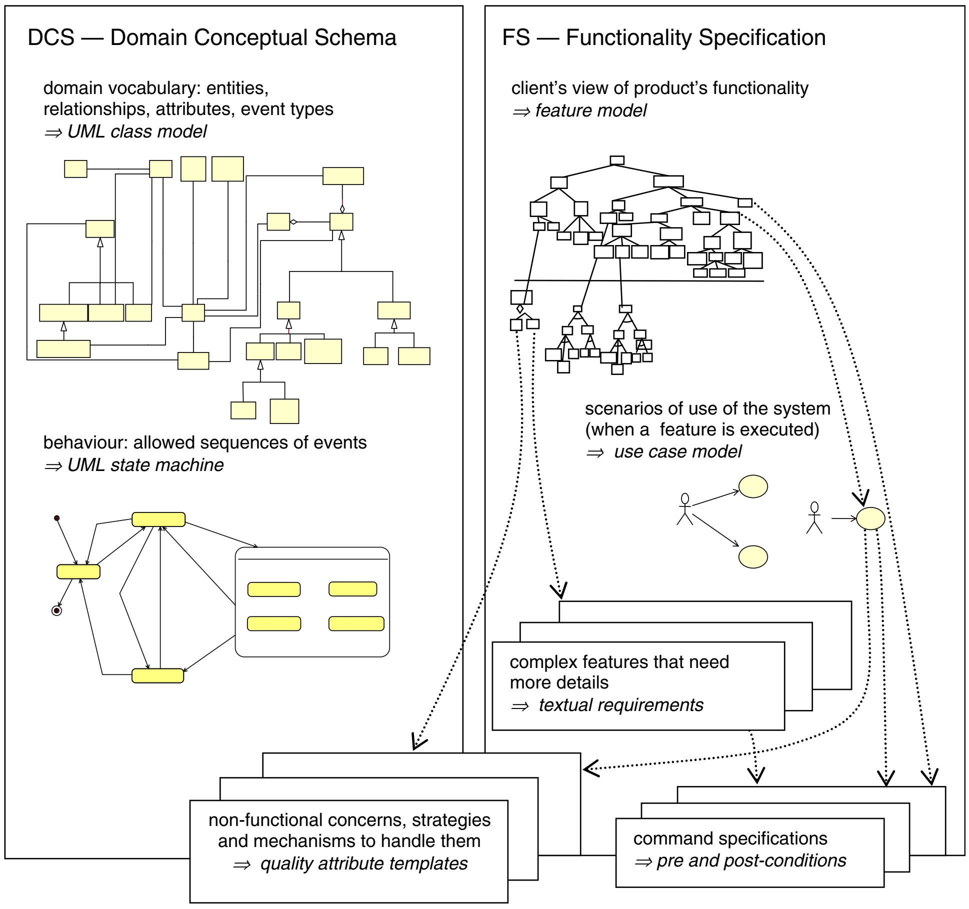

DCS — Domain Conceptual Schema
domain vocabulary: entities, relationships, attributes, event types
⇒ UML class model
behaviour: allowed sequences of events
⇒ UML state machine
FS — Functionality Specification
client's view of product's functionality
⇒ feature model
scenarios of use of the system (when a feature is executed)
⇒ use case model
complex features that need more details
⇒ textual requirements
non-functional concerns, strategies and mechanisms to handle them
⇒ quality attribute templates
command specifications
⇒ pre and post-conditions

Fig. 4 Models used in the TOS domain analysis. *Dotted lines* represent trace relationships

- A generic use case model, describing the interactions between the actors and the system. This model consists of about 20 use cases.
- A collection of textual requirements, which detail features which are especially complex, and which involve an amount of information that cannot be suitably included in the template of the feature. There are about 550 textual requirements.
- A collection of command specifications, which are pre and post-conditions on the commands used in TOS implementation. There are about 25 commands (mainly calibration and movement commands).

Non-functional concerns have been modelled through *quality attribute templates* [16]. These templates do not only encompass requirements, but also help to link requirements and architectural design. One part of these templates corresponds to the DCS, when they describe stimuli related to non-functional concerns (called *abstract scenarios* in this context). The other part of these templates corresponds to the FS, when they describe the strategies and mechanisms needed to solve these scenarios. This is why quality attribute templates are shown halfway between the DCS and the FS in Fig. 4. For instance, abstract scenarios are related to adaptability (e.g. describing changes in elements of the system, such as the kind of tool used to do the cleaning, or the use of a new operating system), performance (e.g. describing changes in the tasks processing time), or availability (e.g. dealing with communications and safe stop availability). When related to safety and security issues these abstract scenarios can be considered threats to the product line, for instance, when the system enters a condition in which it is not safe to execute some commands. The product line has to minimize the risks related to these abstract scenarios. There are about 20 coarse-grained quality attribute templates in the TOS specification.

An essential issue when modelling product lines is variability modelling. In a product line approach, software contains *variation points*, which specify options of behaviour that remain open during core asset development. These variation points have to be instantiated during product development so that the behaviour of the final product is completely specified. In the TOS product line, as we show in next sections, variability is captured in the feature model, in the use cases, and in the quality attribute templates. A sketch of the requirements meta-model designed for the present experience is presented in Fig. 5, whose details are explained in Sects. 5, 6, 7, and 8.

In summary, the process of developing the TOS domain models was performed as follows. Firstly, the DCS was developed to obtain a common understanding of the domain for both the researcher and the CRG. The models

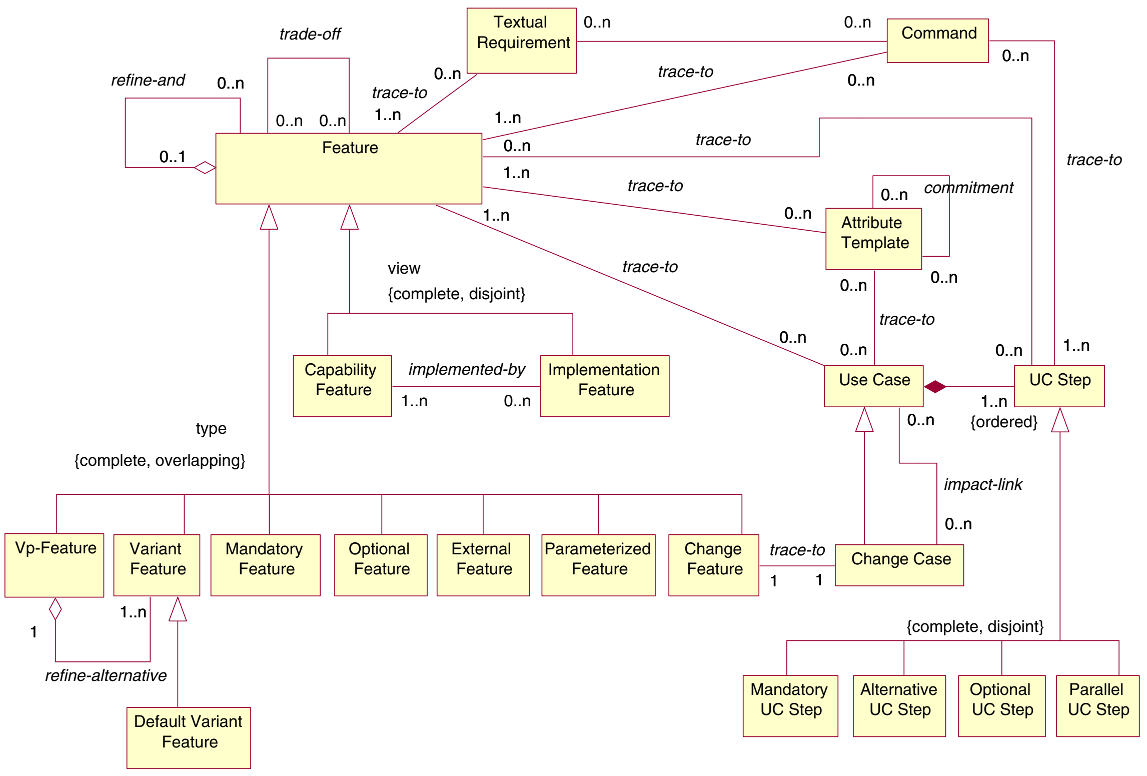


Fig. 5 Requirements meta-model designed for TOS domain analysis

to describe the FS were then chosen and built iteratively. The feature model was chosen first to express the common and variable functionalities of the product line. Some features needed to be enlarged by means of textual requirements, which did not fit properly in the feature templates. Working on the feature model, it was evident that the execution of some product line functionality led to scenarios that needed to be described: a use case model, thus, could be useful, and use case and feature models were performed concurrently. While constructing the use case model, it was quickly realized the necessity to extend *classical* use cases to deal with the variability in the product line. Lastly, quality attribute templates, which were available as a product of the already developed reference architecture, were introduced to deal with non-functional requirements.

5 System capabilities and constraints through features

The capabilities and technology constraints that appear in the products of the product line are specified in the feature model. In this case study, the feature model defined by Kang et al. [14] in FORM has been adopted. FORM has been chosen as the basis of the approach because it is a mature and well-known software product line design method, which has been demonstrated in several case studies (see, for example, [17, 18]). Furthermore, it specifically supports the requirements engineering process, in contrast to other architecture design methods of product lines [19]. The original feature model defined in FORM has been adopted and extended as follows in the next paragraphs.

FORM organizes features in four layers: *capability*, *operating environment*, *domain technology*, and *implementation technique*. With the aim of simplifying the feature model, the four layers have been reduced to only two: *capability* and *implementation*. The latter, therefore, covers the original layers of operating environment, domain technology and implementation technique, which are very close and, in our opinion, sometimes seem to overlap in practice, giving rise to confusion. In this line, for instance, Trigaux and Heymans [20] also criticized the complexity of these four layers. An extract of the feature diagram is shown in Fig. 6, where part of the features related to the services offered by the TOS product line and to the quality aspects are reflected. The relationships between the layers of capability and implementation are specified through *implemented-by* traces, such as the one that shows the types of technology used to do the cleaning, *grit-blasting* and *hydro-blasting*.

Von der Massen and Lichter [21] established the requirements that a notation for variability modelling must satisfy. Subsequently, Trigaux and Heymans [20] extended these requirements with another: the graphical representation of variability, and in particular, of variants, variation points and cardinalities of variation points. In order to satisfy all the requirements and obtain a more expressive notation, the original notation of FORM has been extended with the graphical representation of variation points and cardinalities:

- In line with Griss et al. [22], in the graphical representation of variation points, a feature can represent a variation point (called *vp-feature*), while other features play the role of its variants (*variant features*). For instance, Fig. 6 shows that a relationship *implemented-by* links the feature *hull inspection* with a variant point (*vp-feature*) inside the product line, identified as *camera*, which could be *B/W* or *colour*. Furthermore, a feature can be linked to more than one variation point. For example, Fig. 6 shows how the hull cleaning (*cleaning*) is linked to two variation points: the implied surface *(cleaning area)* and the implied technology (*coating removal technology*).
- With regard to the graphical representation of the cardinalities, we have adopted the approach of Riebisch et al. [23], based on the UML notation, because we think that it is intuitive, simple, and complete. It is complete because it covers all the possible combinations of the cardinalities (min, max) and not only XOR and OR refinements.

We have adopted the concept of *external features* as proposed by Svahnberg et al. [24]. These are features provided by the platform where the system is deployed. They are not part of the system, but are important because they are used by the system, which depends on them. In TOS, for example, the shipyard should provide some specified values of *compressed air supply* and *electricity supply*, with each modelled as an external feature (see Fig. 6). Variability in external features may motivate the addition of software elements to manage this variability.

We have detected the need to add a new type of feature, which we call *change feature*, whose inclusion in the product line is planned for the future, but is not available yet; for example, in the TOS product line, the painting of the ship hull or its welding (Fig. 6).

In addition, we use local and global parameters in the template of FORM's *parameterized features*. The characters $ and @ are used to begin the name of a global and local parameter, respectively. Examples of global parameters are the minimal resolution for a camera in the system and the time of response for a movement of the secondary. Examples of local parameters are the specific resolution of a camera (for instance, the resolution of a colour and digital camera) and the time of response for a movement of an *XYZ table*.

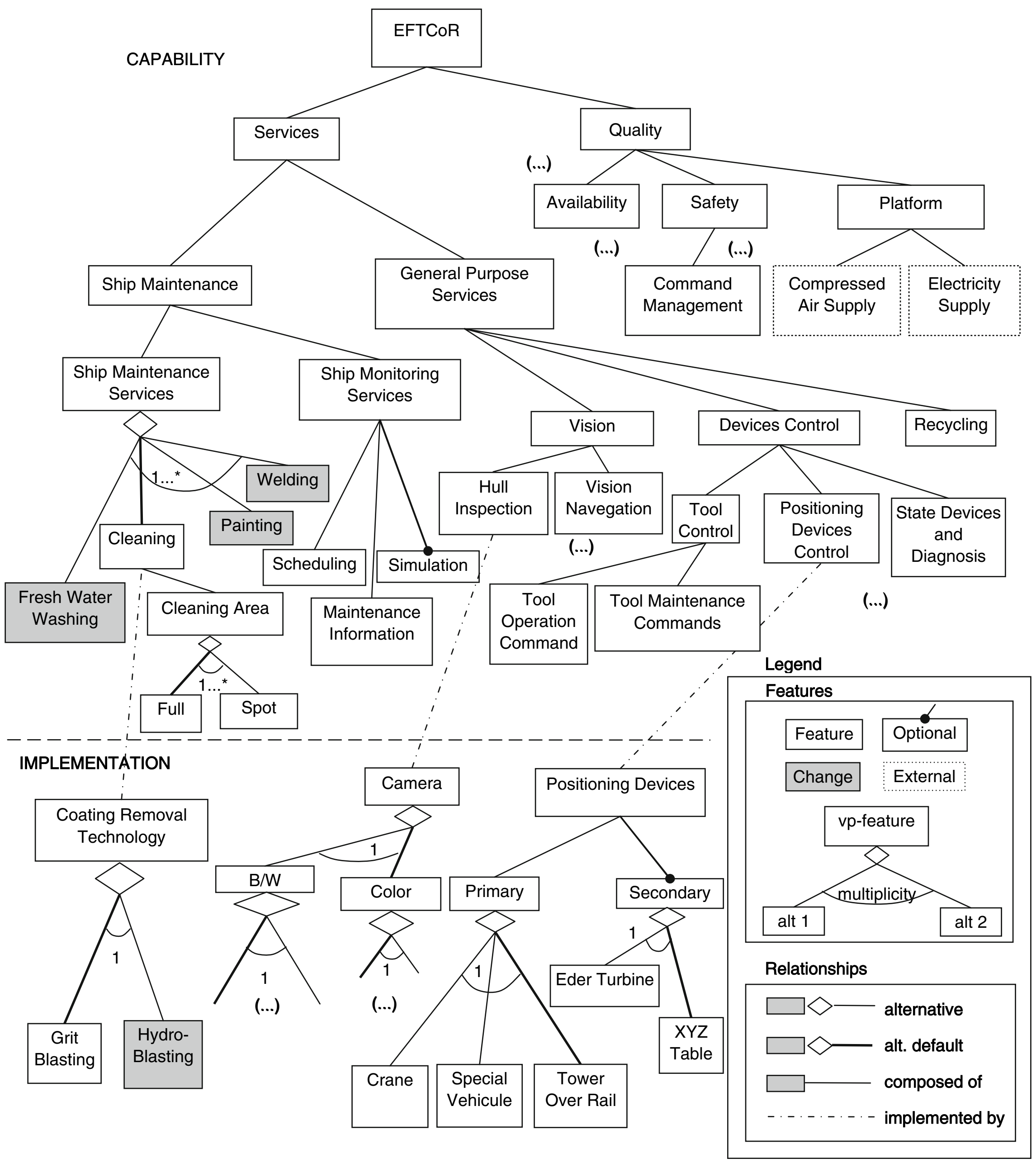


Fig. 6 An excerpt of the feature model

The variability of the product line is reflected in the feature model through the graphical representation of the variation points and their variants, and the optional, external, change, and parameterized features. Complementarily, each feature is described through a template. Figure 7 shows an example of the textual description of the *spot* feature, which is shown in Fig. 6. The textual description of features and the rest of elements in the approach described in Sect. 4 has been carried out by means of the IBM Rational RequisiteWeb^{TM} tool [25], which permits cooperative work through the web and specification of trace relationships.

Composition relationships AND/OR are not enough to express all the possible dependences between features in the model, so a *trade-offs* field (Fig. 7) is used to specify *requires* and *excludes* relationships between features. An example of *requires* is shown in Fig. 7. On the other hand, for instance, *tower over rail excludes eder turbine.* But the trade-off can be more complex than a simple inclusion or exclusion relationship: a feature can sometimes *favour* or

Fig. 7 Textual template for the *spot* feature and some related textual requirements

FEAT Spot (*capability*, *alternative*)

Description: Cleaning performed only on isolated points of the ship hull surface.

Rationale: *Full blasting* is only performed when the hull surface is more than 75% damaged because it implies increasing the costs of abrasive material, painting, staff, etc. Most of the time a partial cleaning (only the deteriorated areas—*spots*) is performed in order to minimize the operation costs. 80% of ships entering the shipyards need this treatment (*spot blasting*).

Composition rules: —

Trade-offs: *requires* (*Primary*, *Secondary*). It is necessary to have primary and secondary positioning systems, since the primary positioning system is not accurate enough by itself.

Trace to: *UC Spot Cleaning, REQ101, REQ102, (…)*

Implements: —

Implemented by: —

END-FEAT

REQ101: The system shall use spot cleaning when less than 75% of the hull is damaged.

REQ102: The spot cleaning implies the use of primary and secondary positioning systems.

REQ103: The system shall force that the movement of the primary positioning system implies the secondary is stopped.

REQ104: The system shall force that the movement of the secondary positioning system implies the primary is stopped.

condition another one. For example, *crane favours XYZ table*, and *special vehicle conditions XYZ table* (because the former has to be adapted specifically).

6 Interaction scenarios through generic use cases

In this case study, the execution of the functionality represented by certain features can be naturally specified as a use case or a combination of use cases: when an actor requires the execution of the functionality represented by a feature with a goal in mind, thus causing a set of interactions with a product in the product line (for example, see the *cleaning* feature in Fig. 6). Traditional use cases are not sufficient to support the variability of a product line, since they describe the actions of an actor when following a certain task while interacting with a particular system. However, the modelling of a product line requires the description of analogous tasks for different products in a product line, that is, *generic* or *product-line* use cases. For the use case diagrams and textual use case descriptions to be suitable for product line modelling, commonality and variability must be integrated and described in them. There is no generally accepted formalism which integrates variability modelling with use cases in order to carry out product line modelling. The approach adopted in this case study is described in this section.

A use case diagram can be used to group the use cases that are involved in the execution of a feature. For example, Fig. 8 shows the use cases related to the cleaning of the hull (*cleaning* feature). This use case diagram is created only for grouping visually the functionality of complex or important features in the product line (such as *cleaning*).

N:M trace relationships are established between features and use cases. For example, the *cleaning* feature is related to several use cases (*full cleaning, spot cleaning, start up, shutdown, calibration,* and *fail recovery*), while the *spot cleaning* use case is related to several features (*cleaning, spot, automatic,* and *teleoperated*). Traceability is textually established between use cases and features and between steps within use cases and features through the feature template (Fig. 7) and the use case template (Fig. 9).

The variation points of the product line are already expressed in the feature model, thus avoiding as much as

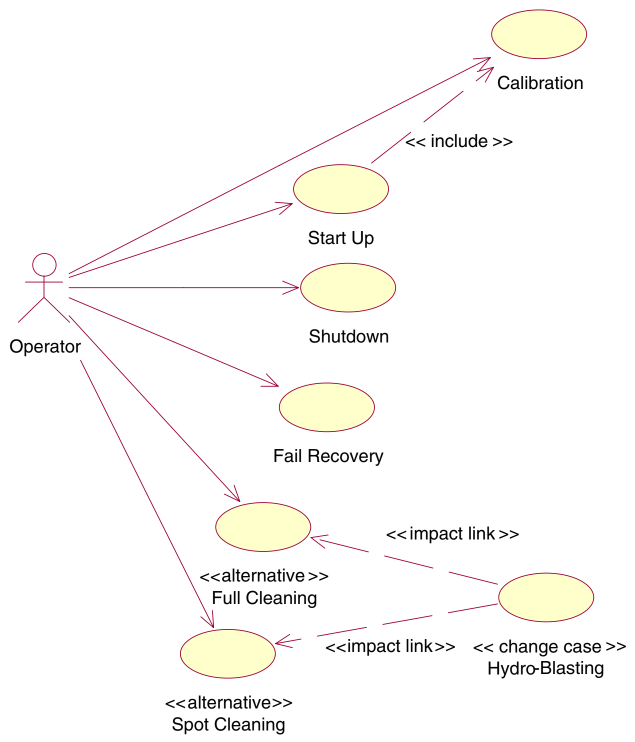


Fig. 8 Use cases related to the hull cleaning

possible an overload of the use case model with the complexity associated to the product line variability. Nevertheless, in order to make use case diagrams more legible, and following Gomaa [26], optional and alternative use cases are labelled with the «optional» and «alternative» stereotypes. In this way, optional and alternative use cases can be easily appreciated visually, although the details of the variability are specified in the feature model. In this manner it can be considered that the use case model is structured by means of trace relationships to the feature model.

The feature model could be considered as a high level interface of the product line. However, there is variability that is intrinsic to use cases in a product line: that associated with the possible variations in the steps of the interaction scenarios, depending on the features selected. This variability has to be captured in the description template of the use case.

After reviewing different approaches to capture the variability of a product line in the use cases [21, 27–30], that of Eriksson et al. [30] has been adopted. This approach has been chosen because it is in line with our focus, and because it proposes the use of:

- *Change cases*, in relation to the possible impact in the use cases of the adoption of future, anticipated extensions of the system, which are still unavailable. These are special use cases, originally proposed by Ecklund et al. [31], which capture possible extensions of the system. Use cases which can be affected by each change case are indicated by means of *impact link* trace relationships. The CRG plans to extend its work to take in systems that possess more functionality than EFT-CoR, and it therefore judged the use of change cases to be promising. For example, Fig. 8 shows a change case, *hydro-blasting*, implying a change in the cleaning technique used (*grit-blasting* until now).
- The modelling of the variability in the description of use cases, using:
  - Local and global parameters.
  - An extended version of the textual description of the flow of events. The steps of the scenario where variation can appear are expressed with a special notation.

Figure 9 shows the description of the *spot cleaning* use case. The literals *alt, opt, loop*, and *par* are used to label the steps. For example, alternative steps *alt* (using the same number) are shown within the *description* field in Fig. 9, evincing that the action can be carried out in a teleoperated or automatic form: one step 3 would be traced to the *teleoperated* feature and the other step 3 to *automatic* (analogously to step 4). Optional steps such as (5) *opt* are also given. In addition, a global variable ($MAX_TIME_START) together with two local variables (@MAX_TIME_TOOL, @MAX_TIME_SAFE_-STOP) have been used to express the maximum response time to certain actions within the use case, within the *budget requirements* column. This column refers to non-functional requirements which are related to their corresponding step. For instance, @MAX_TIME_TOOL refers to the maximum time allowed to activate the cleaning tool (step 4). Finally, the *trace to* column contains the commands which operationalize the use case and the related features.

Like Gomaa [26], we have experienced that the construction of feature and use case models is a concurrent, two-way process: the identification of features pinpoints some that are candidates for development by means of a use case, and the identification of use cases and operationalization of these with TOS commands provides feedback on the feature model.

7 Domain conceptual schema as a visual dictionary

The complex TOS domain was unfamiliar to the researchers so that to start the modelling process a DCS was used to describe the vocabulary of the domain, i.e. the concepts of the problem space and the relationships between them. To build the so-called *domain model*, a conventional use of a UML class diagram was adopted

Use case name: Spot Cleaning

Type: ~~mandatory/~~alternative~~/optional~~

Actors: *(primary)* Operator

Summary: Ship hull cleaning in a specified ship hull area (spot). It can be performed in a teleoperated or automatic way

Trace To: *(quality attributes templates and features traced to the use case)*

Quality attributes templates: Command Management, Stop Mechanisms, Access to the User Interface

Features: Cleaning, Spot

Preconditions: The system is started and calibrated

Postconditions: The chosen ship surface area is cleaned

Open issues: -

Description:

Step	**Actor Action**	**System**	**Budget Requirement**	**Trace to**
1	This use case starts when the operator pushes the cleaning button	The system is started to carry out the cleaning operation	Max. response time is $MAX_TIME_START	
2 *loop*	The operator moves the primary with the aim of reaching the cleaning area of interest	The primary is moved		
3 *loop* *alt*	The operator executes commands to move the tool using the images of the ship hull surface, with the aim of placing it in the cleaning area	The secondary is moved		VP-FEAT Execution Mode: Teleoperated
3 *loop* *alt*	The vision system executes commands of the positioning systems to move the tool, with the aim of placing it in the cleaning area	The secondary is moved		VP-FEAT Execution Mode: Automatic
4 *alt*	The operator pushes the button to activate the tool	The tool is activated for the cleaning	Total response time is @MAX_TIME_TOOL	VP-FEAT Execution Mode: Teleoperated
4 *alt*	The system activates the cleaning tool	The tool is activated for the cleaning	Total response time is @MAX_TIME_TOOL	VP-FEAT Execution Mode: Automatic
5 *opt*	The operator pushes the emergency stop button.	The system stops safely	Total response time is @MAX_TIME_SAFE_STOP	FEAT Emergency Stop
6	The system checks the quality of the cleaning	If it is OK then 7, if not start again, 3		
*7	At any time, the operator pushes the cancellation button	The cleaning stops at that point		

Fig. 9 Fully dressed *spot cleaning* use case

(e.g. see Larman [32]). The DCS was extended with information on the dynamics of the domain event types through a UML state machine. Event types helped to know the states of the system better.

As concluded by Lee et al. [33], in a new domain that is not still mature, the standardization of the domain terminology and the use of standard terms during the analysis can accelerate the feature identification process.

The domain entities involved in the feature templates, textual requirements, use cases, and quality attribute templates, must be specified in the DCS. In addition, some functionality of the TOS systems was first described in the DCS, and included later in the feature tree. However, not all features in the feature model are in the conceptual schema. For instance, low-level features that were discovered during feature model construction (e.g. some implementation features) and most of the intermediate levels of the feature tree were not included in the conceptual model. In the present experience the conceptual model was "frozen" after providing domain understanding, a common vocabulary and a starting point to build the feature model. The conceptual model was not used later except to document the knowledge about the domain graphically.

A part of the class model of the DCS is presented in Fig. 10, including interactions between the RDCU and the other subsystems: *vision subsystem*, *monitoring subsystem*, and *recycling subsystem*. In fact, these subsystems belong to the architecture, but they were included in the DCS for the sake of domain understanding, as long as they were described by the CRG as fixed in their description of the domain. Only some of the attributes are shown in Fig. 10 owing to size and legibility.

The DCS is completed by the event types described through a UML state machine (see Fig. 11). The state machine shows that, when the system starts, it remains in an *idle* state until it is made *operational*. Transitions between states are produced by a *change_of_mode()* signal, with the exception of the transition to *safe stop* which is a result of an alarm. When the alarm is restored, the system will return to *operational* or *idle* depending on the severity of the alarm. The *maintenance* state can be considered as a limited operational state which allows calibration, diagnosis, configuration or apprenticeship operations to be performed with guarantees of safety (for instance, the velocity of the axis and automated commands are limited).

The researcher discussed whether it would be useful to augment the degree of formality in the construction of the DCS to arrive at the definition of a TOS domain ontology. For example, the proposal of Guizzardi et al. [34], which defines a UML profile, could be used to create an ontology of TOS. The use of an ontology-defining language will produce an overload in terms of effort and legibility, and therefore the researcher believed that it should not be used in the absence of evidence as to the benefits that it will confer. A more practical approach has been chosen in the case under study.

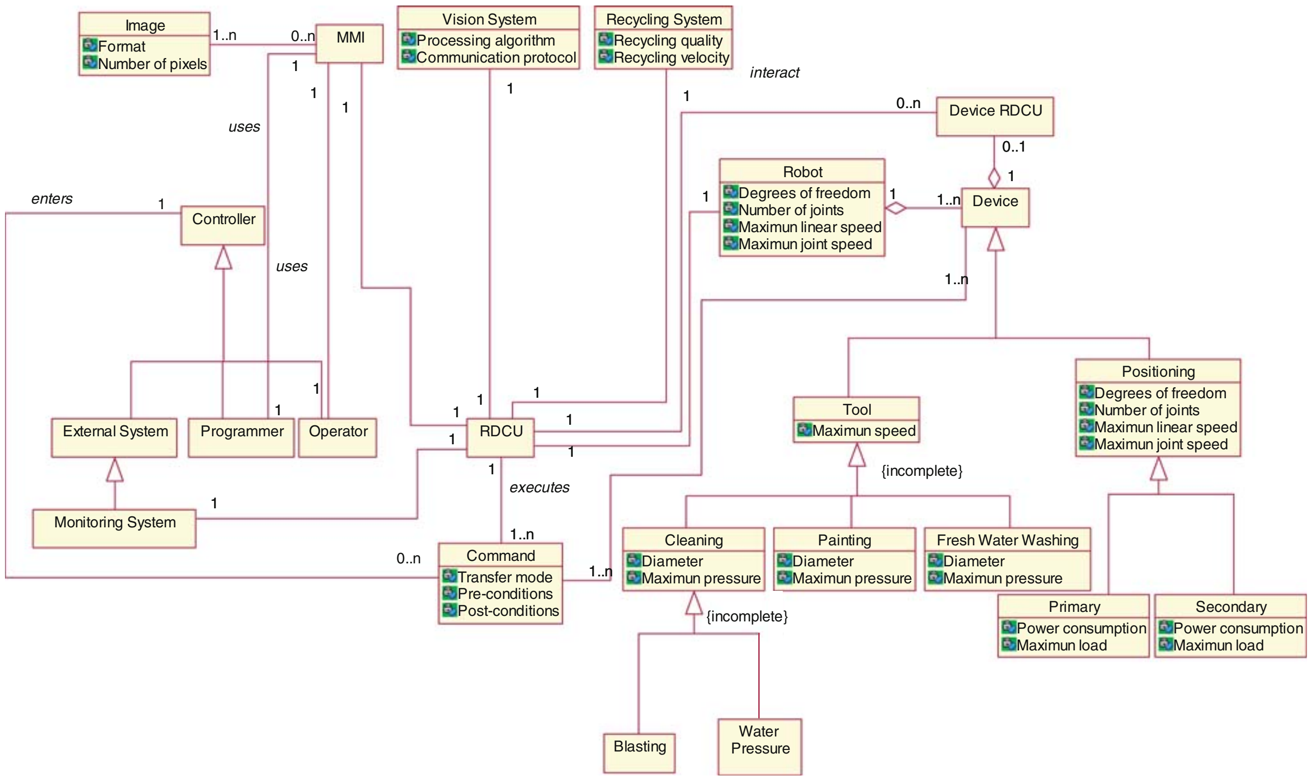


Fig. 10 An extract of the entities of the DCS

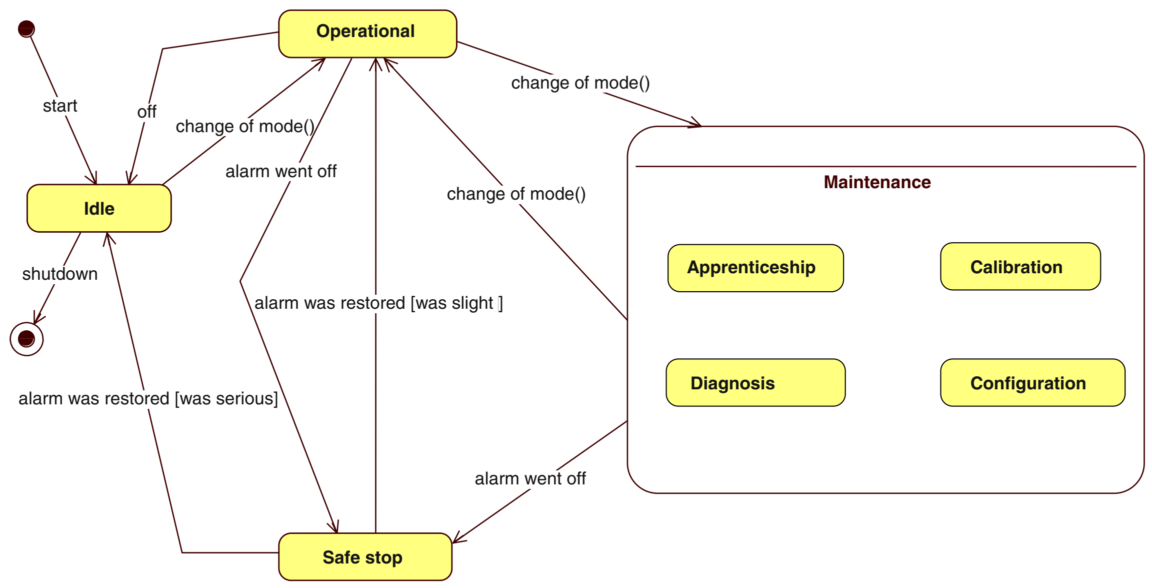


Fig. 11 State chart describing the RDCU (*robotic devices control unit*) changes of mode

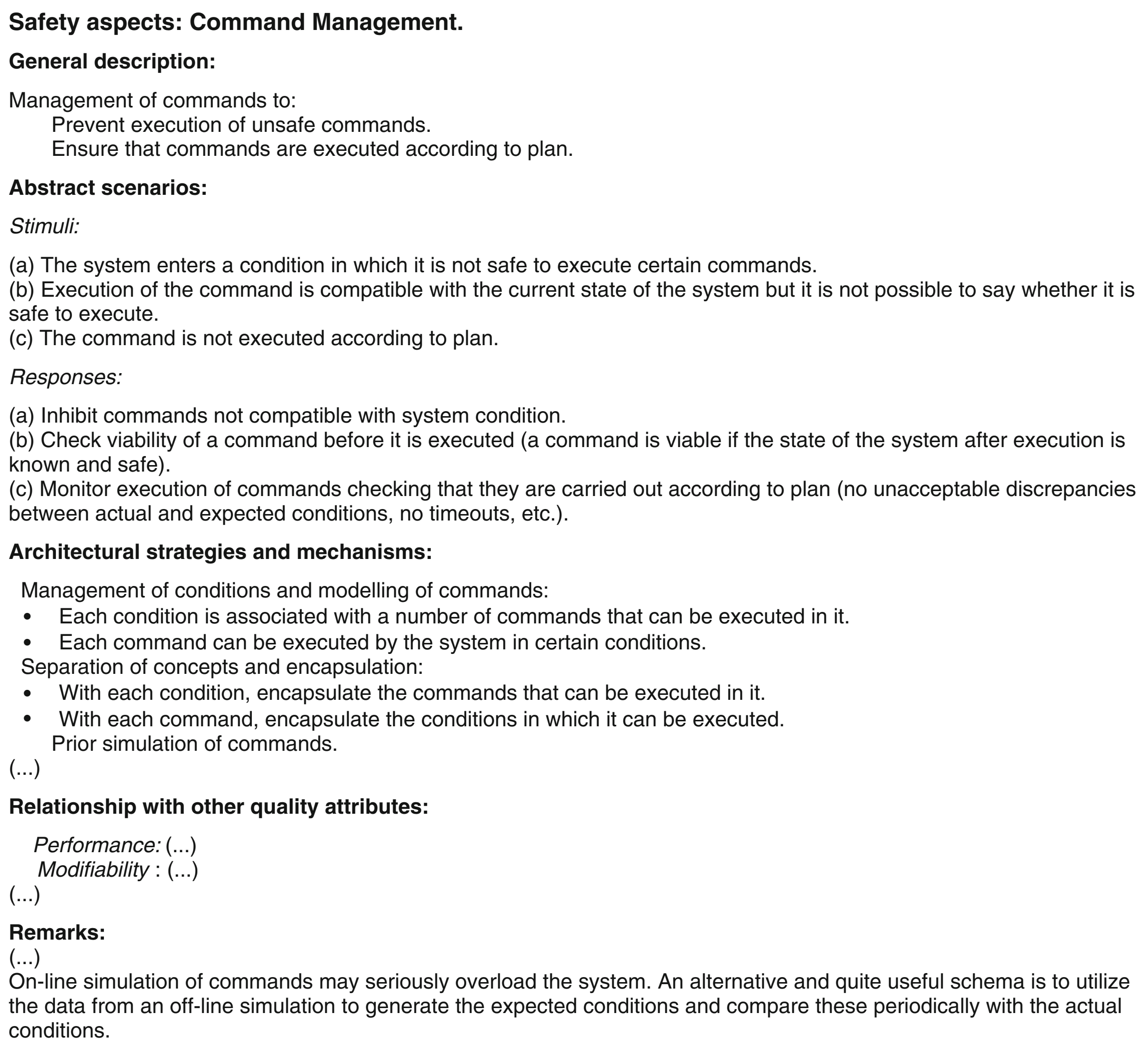
8 Approaching architecture and implementation: quality attributes and commands

Certain emergent, non-functional properties of the system acquire particular importance in the design of a generic architecture for TOS. These *quality attributes* have been modelled following the recommendations of Bass et al. [16], because the CRG had already worked with them in the specification of their reference architecture [1]. As was stated in Sect. 4, the quality attribute templates include a description of the abstract scenarios related to non-functional concerns that the system has to take into account in the product line, and the responses that these should produce. Moreover, possible strategies and mechanisms for solving these abstract scenarios are provided, as well as dependence relationships between the quality attributes. These dependences are more complex than a simple inclusive dependence, as they include commitment relationships—identified by Chung et al. [35].

Quality attribute templates may be seen as clusters which encompass requirements, design strategies and mechanisms related to a particular non-functional aspect. For example, concerning safety in TOS, it is essential to constantly monitor the system's operation, to manage the execution of commands and to provide safe shut-down mechanisms that are activated by the operator or when hazardous situations or serious operating errors are detected. For instance, command management is necessary to prevent the execution of unsafe commands and to ensure that commands are executed according to plan (see Fig. 12). In this figure *stimuli* can be thought of as the *causes* of the product line abstract scenario while *responses* can be seen as the proposed solutions. In the figure response (a) corresponds to stimulus (a), the system must inhibit commands which are incompatible with the system condition for reasons of safety, e.g., it must not move autonomously if it is in maintenance mode. The same correlation can be found between stimulus (b) and response (b), and so on. Next, the Architectural strategies and mechanisms field provides strategies and mechanisms for dealing with or solving these scenarios, e.g., each system status is associated with a number of commands that can be executed in it. In identifying the architectural strategies and mechanisms best suited to satisfying the quality attribute, we are overlapping domain modelling with architecture design.

To sum up, in the present experience non-functional requirements have been specified by means of: (1) *quality features*, which are the features related to non-functional concerns in the feature model [14], (2) quality attribute templates, which are traced to quality features, and (3) use cases, where non-functional requirements are specified by traces to quality attribute templates (field *trace-to*), and in the steps of the use case (field *budget requirement*).

Commands are functional events which are interchanged between subsystems in order to specify which operations one subsystem requires from another. For example, the *Vision_Subsystem* detects an area to be cleaned and thus sends a command *jog_motion(joint_id, angle, position, direction, velocity, acceleration)* to the RDCU (*robotic device control unit*) to force the robot to move to the desired area. TOS systems typically share the same commands between applications (*jog_motion, stop, move_forward*, etc.), although their implementation might be different. In



Safety aspects: Command Management.

General description:

Management of commands to:
Prevent execution of unsafe commands.
Ensure that commands are executed according to plan.

Abstract scenarios:

Stimuli:

(a) The system enters a condition in which it is not safe to execute certain commands.
(b) Execution of the command is compatible with the current state of the system but it is not possible to say whether it is safe to execute.
(c) The command is not executed according to plan.

Responses:

(a) Inhibit commands not compatible with system condition.
(b) Check viability of a command before it is executed (a command is viable if the state of the system after execution is known and safe).
(c) Monitor execution of commands checking that they are carried out according to plan (no unacceptable discrepancies between actual and expected conditions, no timeouts, etc.).

Architectural strategies and mechanisms:

Management of conditions and modelling of commands:
- Each condition is associated with a number of commands that can be executed in it.
- Each command can be executed by the system in certain conditions.

Separation of concepts and encapsulation:
- With each condition, encapsulate the commands that can be executed in it.
- With each command, encapsulate the conditions in which it can be executed.

Prior simulation of commands.
(...)

Relationship with other quality attributes:

Performance: (...)
Modifiability : (...)
(...)

Remarks:
(...)
On-line simulation of commands may seriously overload the system. An alternative and quite useful schema is to utilize the data from an off-line simulation to generate the expected conditions and compare these periodically with the actual conditions.

Fig. 12 Quality attribute template associated to management of commands (extract)

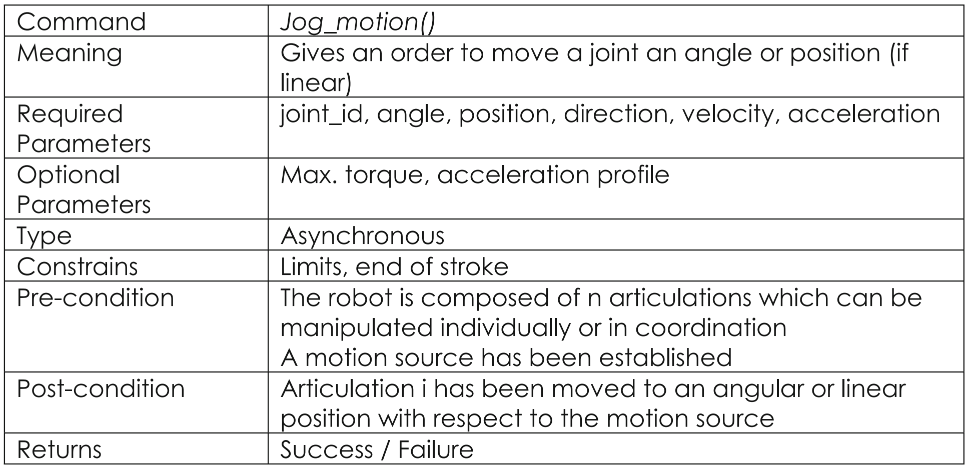
our approach these commands can be accessed directly from the features, from the textual requirements, or from steps in the use cases. TOS commands are described in templates by their meaning, parameters, type, constrains, and pre and post-conditions. Figure 13 presents an extract of the template which defines the main parameters of a *jog_motion* command.

9 Lessons learned

The TOS domain model was defined after constructing a first prototype called GOYA [36]. This prototype was the first robot developed by the CRG for ship maintenance and allowed them to perform an exhaustive study of the TOS domain and to define features, use case models, and quality attribute templates. TOS models were then instantiated in the EFTCoR project for the products shown in Table 1. The knowledge acquired during the development of these EFTCoR systems has enriched the TOS domain model.

Although some members of the CRG participating in the EFTCoR project had previously developed the GOYA prototype, it was difficult to collect representative quantitative data to be used in a comparison because the GOYA project was not developed by taking software engineering concerns into consideration during the requirements specifications, and no data were collected. Thus, we can only provide general data with regard to developing time, people involved, and the number of systems that were built in both projects (see Table 2). In any case, we believe that the qualitative research method chosen, action-research, has been useful because it has driven research towards practical objectives and has helped industry to put the results of the research into practice.

In this section we discuss the main lessons learned, by summarizing the EFTCoR project personnel's experiences



Command	*Jog_motion()*
Meaning	Gives an order to move a joint an angle or position (if linear)
Required Parameters	joint_id, angle, position, direction, velocity, acceleration
Optional Parameters	Max. torque, acceleration profile
Type	Asynchronous
Constrains	Limits, end of stroke
Pre-condition	The robot is composed of n articulations which can be manipulated individually or in coordination A motion source has been established
Post-condition	Articulation i has been moved to an angular or linear position with respect to the motion source
Returns	Success / Failure

Fig. 13 Specification for jog motion command (extract)

Table 1 EFTCoR products for which the models have been instantiated

Product ID.	Cleaning operation and ship hull area	Primary system	Secondary system	Tool
P1-SV	Spotting (vertical surfaces)	Vertical towers	*XYZ* table	Cleaning head with residues confinement
P2-SB	Spotting (bottoms)	Scissor crane	*XYZ* table	Cleaning head with residues confinement
P3-FV	Full blasting (vertical surfaces)	Vertical towers	–	Cleaning based on turbines
P4-FS	Full blasting (shaped surfaces)	Vertical towers	–	Cleaning head with residues confinement

Table 2 Comparison between GOYA and EFTCoR development times and systems

Project	Duration (years)	Fixed personnel (people)	Auxiliary personnel (people)	Systems built
GOYA	2	8	4	1 prototype
EFTCoR	3	12	4	4 systems

in using the TOS domain model. These are organized into groups in order to ease reading. The impressions of the CRG when adopting the TOS domain model have been collected and evaluated through interviews, observations, and group discussions. In addition, in order to quantify their experiences and opinions, the personnel implied in TOS development (ten people) filled out a 20-point questionnaire.

9.1 FORM features are useful in the structuring of a reusable requirements catalogue and speed up the search for requirements in the catalogue

Prior to this experience, the researcher had worked on the reuse of textual requirements in the domains of personal data protection [37] and security [38], but had not used FORM features. The researcher therefore knew from these experiences that a reusable requirements catalogue in natural language for a broad domain may be sufficiently precise and correct, but that it may sometimes be hard to handle since it is made up of long lists of textual requirements which are arranged in sections in a hierarchy of documents. For instance, it is at times difficult to locate the requirements that are related to a certain concern. The requirements catalogue in the TOS domain, however, has been constructed directly on the basis of FORM features. The use of features leads to more rapid reasoning in the definition of a product in the product line than the use of textual requirements only. When specifying a product in the product line, it is possible to browse the problem space by means of the *decision space* defined by features, and to select those required, more quickly than by going through long lists of textual requirements.

Regarding the CRG, almost all the people involved in the project who had previous knowledge of the domain since they had already worked on the GOYA prototype found the feature model useful and easy to use in the definition of a new product, and preferred to have a graphical representation of features rather than a textual description in order to define the main features of a new product quickly. Ninety percent of those polled in EFTCoR found the graphical representation the most appropriate, but most of them also suggested that a software tool to manage the model would be useful, especially when the model is bigger.

Like Griss et al. [22], the CRG decided that it was necessary to include *views* in the feature model to deal with complexity, and so two levels of abstraction were included—*capability* and *implementation*—. These have proved useful for dealing with the complexity of the model while simplifying the four levels of FORM features. The experience in the EFTCoR project reflects that 80% of people positively appreciate using two views to separate capability from implementation.

9.2 FORM features are useful to specify variability in requirements

We can again compare this experience with the work of the researcher in the domains of personal data protection [37] and security [38]. No variability model was explicitly developed in the aforementioned works. Variability in natural language requirements was expressed by means of natural language in the requirements text, parameterized requirements, and inclusive and exclusive traces between requirements (in short, ReqA *includes* ReqB if for ReqA to be satisfied, ReqB must also be satisfied; ReqA *excludes* ReqB if ReqA cannot be satisfied in the case that ReqB is satisfied). In these early experiences variability is consequently broadened around the whole requirements specification.

After the experience in the TOS domain, the researcher believes that the feature model contributes towards a clearer expression of the variability in the product line than the two previous approaches. The feature model plays the role of a *variability model* which encompasses the variability of the product line that is relevant for an eventual purchaser of a product of the product line: the feature model shows the variability graphically and it extends the variability constructs which are present in the two previous approaches. Graphical representation of the variation points in the feature model (*vp-features*), which extends the FORM notation, has been useful in making the decisions that have to be taken in the instantiation of the family explicit. However, using a more expressive notation to denote cardinalities has only proved useful in showing XOR and OR nodes.

During the reflection stage of the present experience (Sect. 3), it was found that variability management as performed with feature and use case models can be improved by using decision models like those proposed in PuLSE^{TM} (*product line software engineering*) [39]. These models list the alternatives for each variation point and show the consequences of choosing each alternative. Decision models are particularly useful in large-scale projects. In a large feature model the clear establishment of the different products making up the product line is not a trivial task.

9.3 Direct traces from requirements to commands help to reuse code

The CRG believe that traceability from the product line requirements to the TOS commands is important. To this end, features, use cases, and quality attribute templates have been operationalized with TOS commands. These are very common commands, both in the TOS family and in other robots, and it would be extremely useful to be able to reuse them. For instance, some commands are *move_primary_to(), move_secondary_to(), execute_sequence(), jog_x(),* and *lookfor_spot()*. When instantiating a new product of the family, we know from the feature model which capabilities the new product needs—for example, *vision navigation*. It is very useful to have a trace from that feature to the command *lookfor_spot(),* which includes algorithms that can be directly reused. The developer thus knows from the outset which commands will be needed.

9.4 All the techniques in the approach should not always be used to model a new product line from scratch

The framework used in the present experience was designed in the context of existing TOS. Reference architectures had already been developed in the TOS product line and quality attribute templates were available. It may be asked whether this framework is suitable to model a new product line from scratch. In this case the researcher believes that if the product line is not especially concerned with safety issues, it might be worthwhile specifying non-functional requirements simply by means of natural language rather than quality attribute templates, which add more information but are, on the other hand, much harder to write. Quality attribute templates are especially useful in safety critical systems such as TOS systems. Furthermore, prior to modelling features and use cases it might be interesting to build a conceptual model quickly only if a number of new concepts in the domain need to be understood. In the present experience the conceptual model has been useful to assist in the development of the feature model. Nevertheless, the researcher now knows the domain well and thus the conceptual schema would not currently be needed, so feature and use case modelling might be built directly.

Some of the main impressions with regard to the usefulness of the approach are summarized in Table 3. It is noteworthy that personnel with previous knowledge of the domain are more enthusiastic about the approach presented in this paper, probably because many of them worked on the GOYA prototype and are aware of the amplitude and complexity of the TOS domain and the difficulties of developing such systems without a disciplined approach.

10 Validity of the results

This section provides a critical vision of the validity of the results obtained in the present experience, and in particular

Table 3 Main conclusions extracted from questionnaires

	Feature model	Use case model	Attribute templates	Reuse of the domain model
Personnel with previous knowledge of the domain	83% think that it is the most useful to understand the domain and to define new products	33% think that it should be extended (more than 30%) for new developments	33% think that they are difficult to use and that they do not help to explain the domain	83% think that it can be reused in future developments (at least at 75%)
Personnel without previous knowledge of the domain	50% think that it is more useful than use cases to understand the domain and to define new products	75% think that it should be extended (more than 30%) for new developments	75% think that they are difficult to use and that they do not help to explain the domain	50% think that it can be reused in future developments (at least at 75%)

of the role played by A-R. The weaknesses of A-R arise from misunderstandings in its application rather than from its nature in itself. Baskerville and Wood-Harper [40] and Baskerville [41] have provide an in-depth discussion of the risks associated with A-R and how they can be handled. Two problems that have sometimes arisen in the present experience are the following:

- The absence of a method which will permit researchers and practitioners to use and conceive A-R.
- The absence of a detailed research process model which shows the steps to follow in A-R.

These problems jeopardize the rigour in the research process. There is a risk in that researchers and practitioners *forget* the framework of the research, and err in rigorously defining the steps of planning, action, observation, and reflection. This has happened at times in the present experience. If this experience were to begin again, the researcher would define the objectives of the A-R cycles more strictly, and smaller cycles with more limited objectives would be designed. It should be taken into account that A-R was used in the present experience for the first time by both the researcher and the CRG. A-R has been shown to be a method which makes it possible to establish a simple framework for the research, although in practice the application scheme shown in Fig. 3 has proved to be too inflexible if strictly followed, and the joint work has proceeded with constant turn-backs and feedbacks. In the present experience, for instance, it was not possible to determine all the models that were necessary to carry out the modelling (*planning*) until this had commenced (*action*). In order to avoid the aforementioned problems and to improve the rigour of the application of A-R, Estay and Pastor [42] propose driving the research through a project management perspective, by generating a project structure that encompasses the main elements of A-R. Research and project are equivalent concepts for these authors.

A further problem in the application of A-R occurs when the researcher adopts a role of mere consultant in the process, sometimes by contract. When this happens, the researcher has a limited capacity to change the practice. In the present experience, however, researchers and CRG staff were able to communicate through a common *software engineering language*, and the CRG did not impose limits on changes in the practice.

The conclusions drawn from the present experience would be more valuable if quantitative data had been collected. The improvement in the EFTCoR project in relation to GOYA is evident from Table 2 (three more products built with only four more people and one more year), but it should be taken into account that the figures in EFTCoR are better not only due to the TOS domain model, but also due to the previous experience of six members of the development team, who had previously worked on GOYA.

11 Related work

On the basis of their work in aviation engine control systems, Lam et al. [43] have identified ten key questions which can benefit an approximation to requirements reuse. The present experience leads us to agree with most of the steps proposed by Lam et al.: (1) to be reusable, the specification of requirements needs to be generalized, but to improve reuse the details also need to be considered, (2) it is desirable to identify product families in order to maximize reuse, (3) reuse techniques need to be evaluated in terms of their impact on the process, and we have therefore been working with simple, well-known approximations, (4) knowledge of the domain should be used to organize reusable artefacts, as for instance we do by means of the feature model, the use case model and the quality attribute templates, (5) inclusive trace relationships need to be introduced between requirements, so “requirement B is only possible if requirement A is true”, as we do by means of the hierarchical decomposition relationship in the feature model, (6) the context of reuse needs to be made explicit so as to avoid its wrong use, to which end we use external features and trade-off relationships between features, and (7) it is recommended to assess the impact of requirements reuse on other development processes such as design. We are not questioning the utility of the three remaining key questions mentioned by Lam et al.; however, we must say that in the present experience they did not prove relevant, (8) it was not necessary to use the proposed method by Lam et al. to identify parameterized requirements, which were identified in an *ad hoc* manner in the present experience, (9) it was not necessary to identify pluggable requirement parts which could have been used to improve the definition and instantiation of the parameterized parts of the requirements, and lastly (10) we were unable to assess the statement that "parts of the requirements engineering process are also reusable", since the process as such was not modelled. In fact the present experience was guided more by those products described in Sect. 4 than by a particular process model.

As we can see, for Lam et al. [43] one of the key questions for improving reuse is the adoption of a product line approach. Clements and Northrop [4] have described the specific practices that are of use in the area of requirements engineering for the development of product lines, and these were put into practice: (a) domain analysis techniques, (b) feature modelling, (c) use case modelling, (d) change case modelling, and (e) traceability of requirements to associated work products. The only question mentioned by Clements and Northrop that was not followed up was the modelling of stakeholder views, as in the present experience we only worked with one *requirements supplier*, the CRG, who are the main developers of the products and who had a clear idea of the product line requirements.

Several authors, such as Pohl et al. [6] and Bayer et al. [44] propose the use of a *variability model* to express the variability of the product line. This variability model includes the variation points and alternatives of the product line and is independent of the other development models. The basic idea is that the alternatives in the variability model are traced to subsets of the development models. For example, an alternative could be traced to certain features and use cases at the requirements level. In particular, Pohl et al. [6] propose the use of an *orthogonal variability model* which solves some limitations of the feature models as originally proposed by FODA/FORM. Pohl et al. [6] report two shortcomings in these diagrams: (1) There is an inability to distinguish between alternative features that are common to all applications (and which are therefore a commonality of the product line) and alternative features that can be selected separately for a specific product. In the TOS domain all alternative features have to be selected in development time. (2) The feature tree lacks a grouping mechanism which would allow arbitrary features to be assigned to a particular variant. The developers might wish to offer several variants, which are split across different branches of the feature tree. Restructuring the feature tree according to the grouping is not always a viable option, since the original decomposition of features is then lost. We have not needed these *package features* in the TOS product line, and they have therefore not been handled in our approach, but we do not question their general applicability.

12 Conclusions and future work

We have presented an experience in analysis of the TOS product line, with a view to enabling reuse of requirements in this domain:

- We have presented a model of the TOS domain which is designed to raise the level of abstraction with which stakeholders can reason about this product line.
- We have shown the main lessons learned from this experience in TOS modelling, which was designed using a qualitative research method, A-R.

To the best of our knowledge, there is no specific proposal to tackle TOS requirements management. To construct the TOS domain model, then, we performed a critical analysis of the state of the art in product line requirements engineering, in the course of which we selected and enriched existing techniques, the applicability of which we have experienced in this domain. We have selected techniques that are well-known in domain analysis to render the proposal more accessible.

The impressions of the CRG when adopting the TOS domain model were collected and evaluated through interviews, observations, group discussions, and questionnaires. They reflect that CRG experienced personnel found the adoption of software engineering techniques useful. The CRG considers the present experience a success in that the requirements documentation generated for the product line lends it added value in terms of dissemination to both customers and developers, since it is not usually developed in teleoperated systems.

We believe that the use of domain analysis and requirements as presented in this paper can be extrapolated to other teleoperated systems: the requirements models could be different, but we think that the approach is valid for modelling them, because they share a basic set of commands and the scenarios of use are similar.

We believe that the approach presented in this paper would be more useful if a tool was available to generate requirements documentation automatically from the analysis models. The idea is to define and automate the mechanisms for transformation of the analysis models into textual requirements that could be labelled with attributes such as their priority or degree of accomplishment. A software product line tool, GEARS [45], already permits the generation of the requirements documentation based on the features selected during product derivation. The documentation that GEARS generates is a snapshot of the derivation of a product in the product family: the requirements document cannot evolve together with the software product line models. Our idea is to go one step further, which implies providing two synchronized *views* of the software product line: the *models view* and the *textual view*. In an iterative and incremental development, the textual view would simplify the management of the requirements of the project, and it could help the client to discover the state of the project and to fix the terms of the contract.

Acknowledgments Partially financed by the CICYT (*Science and Technology Joint Committee*), Spanish Ministry of Science and Technology (TIN2006-15175-C05-02 and TIN2006-15175-C05-03). We would like to thank Prof. Alan M. Davis for his selfless help in the revision of the manuscript. Of course any remaining fault is ours.

References

1. Fernández C, Iborra A, Álvarez B, Pastor JA, Sánchez P, Fernández-Meroño JM, Ortega N (2005) Co-operative robots in the ship repair industry. IEEE Rob Automation Mag 12(3):65–77
2. Ortiz FP JA, Alvarez B, Iborra A, Ortega N, Rodriguez D, Conesa C (2007) Robots for hull ship cleaning. In: IEEE international symposium on industrial electronics (ISIE 2007). Vigo, pp 2077–2082
3. Álvarez B, Sánchez P, Pastor JA, Ortiz F (2006) An architectural framework for modeling teleoperated service robots. ROBOTICA—Int J Inf Educ Res Rob Artif Intell 24:411–418
4. Clements P, Northrop L (2002) Software product lines. Practices and patterns. SEI series in software engineering. Addison-Wesley, Boston
5. van der Linden F, Schmid K, Rommes E (2007) Software product lines in action. The best industrial practice in product line engineering. Springer, Berlin
6. Pohl K, Böckle G, van der Linden F (2005) Software product line engineering. Foundations, principles and techniques. Springer, Berlin
7. Käkölä T, Dueñas JC (2006) Software product lines. Research issues in engineering and management. Springer, Berlin
8. Rine DC, Nada N (2000) An empirical study of a software reuse reference model. Inf Softw Technol 42(1):47–65
9. Glass RL, Vessey I, Ramesh V (2002) Research in software engineering: an analysis of the literature. Inf Softw Technol 44(8):491–506
10. Baskerville RL (1999) Investigating information systems with action research. Comm Assoc Inf Syst 2(19):1–31 (article no 4)
11. EFTCoR (2005) Environmentally friendly and cost-effective technology for coating removal. In: 5th framework programme, European Community, subprogram growth reference GRD2-2001-50004. http://www.dsie.upct.es
12. Olivé A (2004) On the role of conceptual schemas in information systems development. In: Ada-Europe. LNCS 3063. 2004. Springer, Berlin, Heidelberg, Palma de Mallorca, Spain, pp 16–34
13. Kang K, Cohen S, Hess J, Novak W, Peterson A (1990) Feature-oriented domain analysis (FODA) feasibility study, technical report CMU/SEI-90-TR-21, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University
14. Kang KC, Kim S, Lee J, Kim K, Kim GJ, Shin E (1998) FORM: a feature-oriented reuse method with domain-specific reference architectures. Annals Softw Eng 5(5):143–168
15. Chastek G, Donohoe P, Kang K, Thiel S (2001) Product line analysis: a practical introduction, technical report CMU/SEI-2001-TR-001, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University
16. Bass L, Klein M, Bachmann F (2000) Quality attribute design primitives, technical report CMU/SEI-2000-TV-017, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University
17. Kang KC, Kim S, Lee J, Lee K (1999) Feature-oriented engineering of PBX software for adaptability and reusability. Softw Pract Exp 29(10):875–896
18. Lee K, Kang KC, Chae W, Choi BW (2000) Feature-based approach to object-oriented engineering of applications for reuse. Softw Pract Exp 30(9):1025–1046
19. Matinlassi M (2004) Comparison of software product line architecture design methods: COPA, FAST, FORM, KobrA and QADA. In: 26th international conference on software engineering (ICSE 2004). Edinburgh, pp 127–136
20. Trigaux J-C, Heymans P (2003) Modelling variability requirements in software product lines: a comparative survey. Computer Science Institute. University of Namur, Namur
21. Von der Massen T, Lichter H (2002) Modeling variability by UML use case diagrams. In: International workshop on requirements engineering for product lines (REPL 2002). Essen, pp 19–25
22. Griss M, Favaro J, d'Alessandro M (1998) Integrating feature modeling with the RSEB. In: 5th international conference on software reuse 1998. Vancouver, Canada, pp 76–85
23. Riebisch M, Böllert K, Streitferdt D, Philipow I (2002) Extending feature diagrams with UML multiplicities. In: 6th conference on integrated design and process technology (IDPT 2002). Pasadena
24. Svahnberg M, van Gurp J, Bosch J (2005) A taxonomy of variability realization techniques. Softw Pract Exp 35(8):705–754
25. RequisiteWeb (2008) IBM Rational RequisiteWeb. http://www-01.ibm.com/software/rational/
26. Gomaa H (2005) Designing software product lines with UML: from use cases to pattern-based software architectures. Addison-Wesley, Boston
27. Gomaa H, Shin M (2002) Multiple-view meta-modeling of software product lines. In: 8th international conference on engineering of complex computer systems 2002, pp 238–246
28. John I, Muthig D (2002) Tailoring use cases for product line modelling. In: International workshop on requirements engineering for product lines (REPL 2002). Essen, pp 26–32
29. Halmans G, Pohl K (2003) Communicating the variability of a software-product family to customers. Softw Syst Model 2:15–36
30. Eriksson M, Börstler J, Borg K (2004) Marrying features and use cases for product line requirements modeling of embeded systems. In: 4th conference on software engineering research and practice in Sweden (SERPS 2004)
31. Ecklund E, Delcambre L, Freiling M (1996) Change cases: use cases that identify future requirements. ACM SIGPLAN Notices 31(10):342–358
32. Larman C (2005) Applying UML and patterns. 3rd edn, Prentice-Hall, Upper Saddle River
33. Lee K, Kang KC, Lee J (2002) Concepts and guidelines of feature modeling for product line software engineering. In: 7th international conference on software reuse. LNCS 2319. London, pp 62–77

34. Guizzardi G, Gerd W, Guarino N, van Sinderen M (2004) An ontologically well-founded profile for UML conceptual models. In: 16th international conference on advanced information system engineering (CAiSE), LNCS 3084. Riga, pp 112–126
35. Chung L, Nixon BA, Yu E, Mylopoulos J (2000) Non-functional requirements in software engineering. The Kluwer international series in software engineering. Kluwer, Boston
36. Ortiz F, Iborra A, Marín F, Álvarez B, Fernández-Meroño JM (2000) GOYA: a teleoperated system for blasting applied to ships maintenance. In: 3rd international conference on climbing and walking robots 2000. Madrid, pp 835–846
37. Toval A, Olmos A, Piattini M (2002) Legal requirements reuse: a critical success factor for requirements quality and personal data protection. In: IEEE international joint conference on requirements engineering (ICRE 2002 and RE 2002). IEEE Computer Press, Essen, pp 9–13
38. Toval A, Nicolás J, Moros B, García F (2002) Requirements reuse for improving information systems security: a practitioner's approach. Requir Eng 6(4):205–219
39. PuLSE (2008) PuLSE (Product Line Software Engineering). http://www.iese.fraunhofer.de/Pulse/Bibliography/
40. Baskerville RL, Wood-Harper AT (1996) A critical perspective on action research. J Inf Tech 11:235–246
41. Barkerville R (2001) Conducting Action Research: High Risk and High Reward in Theory and Practice. In: Trauth E (ed) Qualitative research in information systems. Idea Group Publishing, Hershey, pp 192–218
42. Estay C, Pastor J (2000) Improving action research in information systems with project management. In: 2000 Americas conference on information system 2000. Long Beach, pp 1558–1561
43. Lam W, McDermid JA, Vickers AJ (1997) Ten steps towards systematic requirements reuse. Requir Eng 2(2):102–113
44. Bayer J, Gerard S, Haugen O, Mansell J, Moller-Pedersen B, Oldevik J, Tessier P, Thibault J-P, Widen T (2006) Consolidated product line variability modeling. In: Käköla T, Dueñas JC (eds) Software product lines. Research issues in engineering and management. Springer, Berlin, pp 195–241
45. BigLever-Software (2008) GEARS—software product line engineering tool and framework. http://www.biglever.com/solution/product.html