Experience with Designing a Requirements and Architecture Management Tool

Jun Han
School of Network Computing
Monash University, McMahons Road
Frankston, Victoria 3199, Australia
jhan@monash.edu.au

Abstract

Effective tool support is much needed for the tedious and error-prone task of managing system requirements and system architectures. With the primary objective of providing practical support for software engineers, we have developed a tool for managing system requirements, system architectures and their traceability, which is being used in real-world industrial projects. The tool is based on a well considered information model of system requirements and architectures, and embodies a set of document templates providing guidance for software engineers. In this paper, we report on our experience in designing and improving the tool. In particular, we highlight a number of case studies that played a significant role in formulating the information model and document templates, and provide an assessment of the tool relative to existing practice.

1. Introduction

Management of system requirements, system architectures and the traceability between them is a critical task of system development and evolution. The requirements for a system are the basis of planning, developing, evolving and using the system. The system architecture provides the blueprint or vision for the system's design. The traceability between the system requirements and the system architecture is the key to test whether the requirements are met by the architecture design. In the light of changes to systems, the management of system requirements, system architectures and their traceability has even a greater role to play. It facilitates analysis of how a new or changed requirement will affect the system design and how an architectural design decision will impact on the system's functionality and quality.

In current practice, system requirements are often kept in some monolithic word-processing files. They are difficult to analyse and maintain. The specification of system architectures is usually ad hoc, again hard to analyse and maintain, and difficult to be kept up-to-date. Even with certain tool support, the system requirements and the system architectures are kept separately, and support for their traceability is very limited. Furthermore, most support tools available are either very generic so that only low-level assistance is possible, or too specific by dictating the use of a particular notation.

Aiming to overcome the problems in existing practice and tool support, we have developed a tool, TRAM, for managing system requirements, system architectures and the traceability between them. This tool has *practical usability* as its primary design objective. As such, the tool is equipped with a set of document templates, to provide practical guidance to the user. The document templates are based on an information model for capturing system requirements, system architectures and their traceability. All together, the information model, the document templates and the tool itself provide a practical project start-up kit for requirements and architecture management.

In this paper, we report on our experience in designing TRAM. In particular, we highlight a number of case studies that played a significant role in formulating the information model and document templates, and provide an assessment of the tool relative to existing practice. The paper is organised as follows. We first discuss the design of the tool, including the information model, the document templates and the actual tool implementation. Then, we introduce three case studies of applying the document templates and the tool to real-life systems, and summarise their roles in formulating and refining the tool support. Finally, we present a comparative assessment of the tool relative to existing practice.

2. Tool development

Our tool for managing system requirements and system architectures involves three major aspects:

0-7695-0903-7/00 $10.00 © 2000 IEEE

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

1. a core information model for requirements and architecture management that serves as the basis of formulating the documents templates,
2. a set of document templates for requirements and architecture management, and
3. a support tool.

All these form a project start-up kit, which has been applied to a number of case systems during and after their development for the purpose of refinement and validation. It is currently used in a "live" industrial project at the National Air Traffic Services Ltd (NATS).

2.1. The information model

The core information model for requirements and architecture management sets out to capture the most essential concepts and their relationships concerning requirements and architectures. In doing so, we leverage existing research in three main areas: goal-directed requirements engineering, the world-machine relationship in system engineering, and software architecture design and description.

In goal-directed requirements engineering, the requirements for a system are elicited as goals for the system to achieve. It recognises that the system goals may be stated at different levels of abstraction, from high level business objectives to low level concrete requests. The requirement elicitation and analysis process is such that high level goals are progressively refined or operationalised into lower level goals that are ready for implementation. A representative of goal-directed requirements engineering is the KAOS approach [2].

In [6], Jackson highlighted the need for software engineers to balance the concern between the world, in which the machine they build serves a useful purpose, and the machine itself. When we decide on the requirements for a system, it is primarily about what the system is to achieve or maintain. However, it is also important to know the properties of the system's operating environment that the system must respect. Only with knowing both the system and its environment may we be clear about the boundary or relationship between them and about the requirements for the system.

In recent years, there has been much effort in software architecture research and practice, including architecture description languages such as Darwin [7], architecture styles and case studies [8], architecture practice [1], and our own work on rich specification of software components and architectures [5]. Software architecture design is primarily about devising component-based structures for a system that meet the system requirements. Software engineering practice suggests that requirements engineering and architecture design influence each other during the development and evolution of a system. The relationship and traceability between system requirements and system architecture should be considered for effective requirements management.

The core information model identifies the key concepts and relationships of requirements engineering and architecture design. These concepts and relationships are identified through review of existing literature and analysis of requirements engineering and architecture design practice, and are further refined through application to real-world industrial projects. In the following discussion, we introduce the concepts and relationships of the model.

Stakeholder: The stakeholders of a system are those individuals or organisations who have an interest in the system. They include users, owners, procurers, developers, and so on.

The identification and documentation of stakeholders is key to requirements traceability. Maintaining such traceability is critical to requirements validation and conflict resolution.

Goal: The goals are objectives or desires that the stakeholders *own*, and would like the system to satisfy. The goals generally represent the requirements for the system.

A high-level goal can be *refined* by the combination of a number of lower-level goals, in a recursive manner. Such refinement relationships are to reflect the fact that together with the analysts, stakeholders often express initial requirements in broad and general terms, and then move to identify more detailed and concrete requirements. Retention of the initial and intermediate broad requirements is necessary for the rationale and validation of the detailed requirements. Goal refinement may also be used to resolve conflicting goals and compare refinement alternatives.

Value: Some goals are valued by the stakeholders more than others. The value that a stakeholder gives to a goal represents the level of benefit that achieving the goal will *deliver* to the stakeholder, and highlights the goal's importance relative to other goals in that stakeholder's opinion. We note that the value is related to a stakeholder-goal pair.

In general, values may be drawn from a value system/scheme, or simply represented by statements. They are particularly useful in comparing alternative refinements and resolving conflicts.

Assumption: Assumptions are "indicative" properties of the system's operating environment that the system has to respect or live with. These are fixed in the sense that they are not altered by the system.

Authority: Authorities are those who are in a position and are capable of *asserting* assumptions. They may include management, domain experts and some stakeholders. They may also include static sources such as standards, documentation or similar.

Risk: Not all assumptions can be made with total confi-

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

dence. They may be *subject to* change or their status may be otherwise uncertain. This is the *risk* associated with an assumption. When an assumption is stated, all related risks should be identified and documented.

Interface: The interface defines in concrete terms the boundary between the system and the environment in which the system operates. The interface *makes* the assumptions about the environment visible to the system.

Component: Components are elements comprising the architecture of the system. They can be either pre-existing components or to-be-built components.

The system architecture comprises a hierarchy of components in the sense that a component may have its internal architecture with its own components. Therefore, a component may be *part of* another component.

The system components *conform to* the system interface so that the system can function in its environment through proper interaction with it.

Service: Services are capabilities of the system that are devised to *satisfy* the goals. They are *provided* by system components. In general, a system component provides a number of services and may *require* services provided by other system components. The provided services of a component may be used directly in satisfying the system goals, or used by other components. In general, it is necessary for the system services to *respect all* the assumptions.

Quality of Service (QoS): A service is usually *delivered with* a number of quality properties, such as performance, reliability and security, which are generally referred to as quality-of-service. The specific QoS properties are devised to satisfy certain system goals concerning the quality requirements for the system.

Acceptance Criterion: The acceptance criteria provide the means for establishing the extent to which the services and quality-of-service properties satisfy the goals. In other words, they *test* the satisfaction relationships between services/QoS-properties and goals.

In general, the acceptance criteria should be established according to the goals. The acceptance test cases that practitioners normally use are specific forms of acceptance criteria. In is important to note that the acceptance criteria/cases should include the *extent* of the satisfaction required.

Use Case: A use case presents a type of use scenario for the system, and it *uses/exercises* a number of services. We note that other than validating the system services, the inclusion of use cases in the model also provides a link to UML based system development.

From the practicality point of view, we have tried to keep the information model as concise as possible but without compromising the objective of capturing the essential concepts of requirements engineering and architecture design. The relationships between the concepts are carefully considered, including those between requirements concepts and architecture concepts, so that the necessary traceability is accommodated. The separation between assumptions and goals clearly addresses the relationship between the world (the operating environment) and the machine (the system).

2.2. Templates design

The core information model for requirements and architecture management provides the basis for the content of the envisaged document templates. The document templates capture the information reflected by the concepts and relationships of the information model. They also capture some additional information, including that of the project characteristics and that of the system domain. In general, designing the document templates involves categorising and detailing the information content associated with each concept and relationship, considering additional information necessary to reflect current practice and future evolution.

Representation of concepts and relationships. The information model sets out the basic structure for the document templates. The main information body of a concept is generally represented by a statement or description in the document templates. Upon close examination, however, many concepts and relationships need additional attributes for their clear representation. In the discussions below, we focus on these additional features required by the concepts and relationships for their representation in the documents templates.

The stakeholders and authorities should be about specific individuals, organisations or resources. A stakeholder or authority is given an ID, and has its name, position, organisation and responsibility (relative to the system concerned) recorded.

The goals form a major part of the requirements information for a system. First, we have identified two broad classes of goals, i.e., the goals that the delivered system is to satisfy – *system goals*, and the goals that the system development process is to satisfy – *process goals*. It is also observed that certain system goals are about the system behaviour at run-time while others are about the system as it is designed. Examples of run-time system goals are those about the system's functionality and performance. Examples of design-time goals include those about the system design's maintainability and evolvability. Examples of process goals are goals concerning development standards such as those about validation, verification and documentation, and goals about project budget and duration. In general, a high level goal may concern both run-time and design-time, and even the development process. Only when the goals are sufficiently refined, can clear designation be achieved. We also note that the process goals and design-time system

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 **IEEE**

IEEE COMPUTER SOCIETY

goals do not manifest themselves into system services or QoS properties, and only run-time system goals do.

At finer grained levels, the goals can be classified into various categories, including functionality, capacity/sizing, performance/timing, availability, reliability, safety, security, privacy, operation, adaptability / customisation / portability, system interaction, user interaction, maintainability/evolvability, validation and verification, documentation, and project management. Therefore, the goals have an `categories` attribute in addition to the goal statement.

When a goal is refined into a number of lower level goals, a rationale for the refinement is recorded. The rationale can be for simply detailing the requirements, or for exploring a high level business/design decision.

The assumptions are in general about the properties of the system's operating and development environment. According to their nature, the assumptions can be classified into a number of categories, including system interaction, user interaction, system resources, and standards and regulations. The category of an assumption is recorded as an attribute along side the assumption statement. At the same time, the identified risks related to the assumption are also recorded.

In capturing the component architecture design of a system, it is important to recognise the need to represent the system architecture in a hierarchical manner. That is, the system architecture involves a number of components, and these components may have their internal architectures involving lower-level components, and so on. Therefore, a system component is represented in the context of its immediate enclosing system architectures. For the top-level system and each composite component, there is (1) a component architecture description, identifying the architecture styles/patterns used, the enclosed components, their interactions, and the architectural constraints; (2) specifications of the components; (3) specifications of the inter-component interactions; (4) specifications of the architectural constraints. Note that the system architectures, inter-component interactions and architectural constraints are not explicitly shown in the information model. However, they are essential for clearly defining the system's component architecture.

The architecture description takes the form of a mixture of diagrams and explanations with formal or informal techniques. For a component, whether it is pre-existing or to-be-built is recorded. The component should conform to both the external system interfaces and the internal inter-component interfaces, where appropriate. The specification of an interaction includes the direction(s) and possibly a detailed description of the interaction. The constraints are specified in a chosen notation, informal, semi-formal or formal.

The services are in general operationalised system goals. They are assigned a service category. The service categories are system specific and usually reflect the top level design of the system, i.e., the major system components.

The system interface specification in the document templates may use any chosen notation, e.g., EBNF, with necessary explanations. It also identifies the specific interface's input/output characteristics.

Additional information. In addition to the information suggested by the concepts and relationships of the information model, the document templates also capture some other essential information. One piece of such information is for project identification and management. It involves information about project name, document reference number, dates, author(s), review, authorisation, distribution, revision, and so on. It also includes an overview of the project.

Another piece of information captured by the document templates is about the domain concepts that are relevant to the project. The addition of such information is to reflect the vital importance of understanding the application domain of the system. We have structured the information on domain concepts into two parts. The first part is a glossary of domain concepts, containing the definitions of the relevant domain concepts. The second part is the definition of the system's domain model, which is based on the domain concepts and also captures the relationships between these concepts. It is presented in an appropriate notation such as UML, with necessary explanations. In the current state of practice, the availability and definition of domain models is rare. However, the value of such models is widely acknowledged. To reflect the current practice and prepare for future practice, we have designated the domain model part as optional in the document templates.

We note that an organisation-wide glossary of domain concepts and a corresponding domain model form an important part of the corporate memory. When they are available, the section on domain concepts in the document templates will simply contain references or a subset of the organisation-wide information with necessary additions.

2.3. Tool implementation

We have implemented the document templates in two forms: (1) as HTML document templates, and (2) as templates in the document management tool DOORS, each with its own advantages. We examine below the specific features of the templates in the two forms of implementation.

HTML implementation. In the HTML implementation, the concepts and relationships of the information model have been divided into two document templates: one is named the *System Requirements Document*, and the other

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

the *System Architecture Document*. The System Requirements Document (template) concerns the information about stakeholders, goals, values, acceptance criteria, authorities, assumptions, risks and related relationships. The System Architecture Document (template) contains information about the rest of the concepts and relevant relationships, including system architectures and components, services, QoS properties, system interfaces, and use cases. The division is primarily based on the concepts, regarding whether a concept is primarily a requirements-related concept or an architecture-related concept.

The information concerning each concept is organised in a structured manner. The documents have chapters and sections corresponding to concepts and their categories. The relationships are either embedded in the relevant concepts and/or implemented as hypertext links. As expected, there are relationships between the two documents of any given project. The relationships prove to be a valuable tool for navigating around the documents.

Both document templates include the project identification and management information, while the information concerning domain concepts is included in the System Requirements Document template. In general, the templates design in HTML is a fairly straightforward process.

DOORS implementation. The information model has also been codified into the software document management tool DOORS [9], to set up the document templates. DOORS has a project concept, and within each project there may be many document modules. To implement the document templates in DOORS, all the information as captured in the information model is organised into a DOORS project. Within the project, there is a DOORS document module for each concept. Within each module, the information concerning the concept is structured and organised using the mechanisms provided by DOORS. DOORS' support for cross-module and intra-module fine-grained linking meant that the support for relationships between the concepts (i.e., their instances) is naturally accommodated. The domain concepts are codified in a separate DOORS module while the project identification and management information in another.

The set-up of the different modules in the project is codified as a DOORS template script. This means that the selection of the template script will instantiate the template and create a new project containing all the modules with initial set-ups. The structure for each of the concepts is also codified as a DOORS template script so that we can obtain a new instance of the concept by selecting and instantiating the template script. All the templates are implemented using DOORS' scripting language DXL. Figure 1 shows the project set-up and the templates menu.

As a structured document management tool for systems development, DOORS has a range of features for managing, presenting, subsetting and querying information contained in its project documents. In particular, its support for defining document views and information filters greatly aids the construction, management and analysis of the documents. For example, a view or filter can be easily defined to show only the performance goals in the goal module, or even only the performance goals with given characteristics (e.g., containing a reference to "coarse filtering"). Another filter can be defined to show those system goals that are not refined and are not related to any services or QoS properties. It is obvious that the ability of being able to perform such queries is immensely useful in requirements and architecture management.

Discussions. The two templates implementations have their own pros and cons. Initially, the HTML implementation is used as a testbed for the document templates design, without embarking on template codification work required by the DOORS implementation. The HTML implementation has served this purpose well. In fact, its use is well beyond this initial objective as the following discussion shows.

The HTML templates implementation is light-weight and easily accessible. Without too much work, the basic templates were set up. Because the documents are structured HTML documents, it is very easy for the user to pick it up. Furthermore, the easy interchange between HTML documents and MS WORD documents gives additional accessibility to the HTML templates implementation as many documentation work is carried out in MS WORD in the industry. During the instantiation of the HTML templates, the user may add additional introductory/explanatory information to the documents (and to the templates structure). The HTML document concept is familiar to many people's concept of printable documents, which also leads to consistency between on-line documents and printed documents. HTML's support for hypertext links facilitates easy navigation around the documents.

The HTML templates implementation sets out the overall framework for documents instantiation in a given project context. To actually construct the content of the documents, one needs to instantiate the concept level templates in the primitive "copy-paste-modify" manner. More importantly, the HTML implementation lacks analytical capability for managing the documents, such as that for projecting document views according to given criteria.

As discussed above, we are able to implement directly the information model in DOORS. As such, the templates' DOORS implementation is oriented towards on-line management of requirements and architecture information. It offers navigation capability through its links mechanism (like HTML). In fact, DOORS provides even greater sup-

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

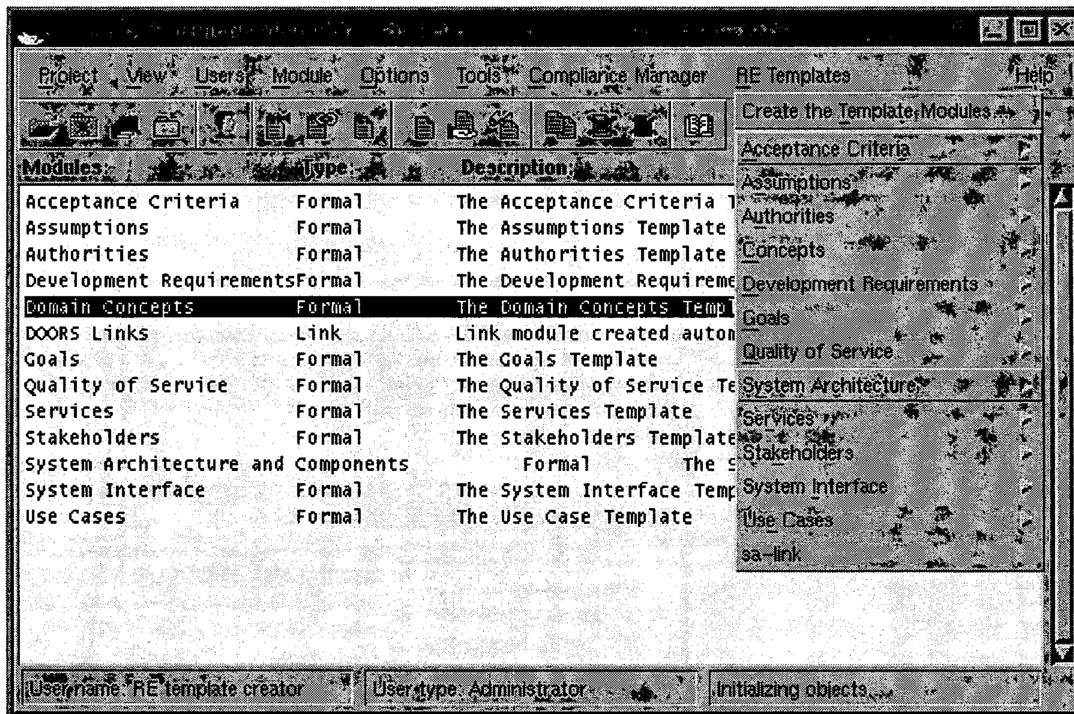

Figure 1. DOORS implementation of document templates

port for navigation. For example, it provides graphical views of fine-grained links between connected modules and graphical views of the internal hierarchical structure of modules, as well as associated manipulation functions. DOORS' support for view definition and information filters provides the templates implementation with much needed analytical capability, as discussed above. For documents content manipulation, one could do "copy-paste-modify" for concept-level templates instantiation in DOORS in a similar manner to the HTML implementation. But with DOORS scripting language, we are able to codify these templates for menu-driven instantiation. DOORS supports document format exchange with external tools like MS WORD. But the document structure imposed by DOORS may be lost in the process as external tools may not support such structures.

Compared with the templates' HTML implementation, the templates' DOORS implementation is relatively "heavy-weight" because it requires the special DOORS tool. The uniform structure dictated by DOORS modules makes it a bit awkward to add extra (free form) explanatory information to the documents. Because of the on-line orientation of the templates' DOORS implementation, producing hard copies of documents requires some special formating and has to conform to DOORS module-based printing.

In general, the templates' HTML implementation provides familiar but basic support for document construction and management, while the DOORS implementation provides advanced support, especially the analytical support, with some extra investment and effort. While both implementations can be used in actual projects, the DOORS implementation is recommended for its advanced features. We note that additional capability can be added to the DOORS implementation for even greater support, e.g., checking of standards compliance [3].

3. Case studies

During and after the development of the core information model and document templates for requirements and architecture management, we have carried out a number of case studies to motivate and refine the model and templates. The three major case studies are Meeting Scheduler, Flight Data Processing (FDP), and Short Term Conflict Alert (STCA). The meeting scheduler system is a well known exemplar system for software engineering research [4]. The remaining two case studies are all based on real-life Air Traffic Control (ATC) systems. Based on our positive experience with the case studies, the information model, document templates, support tool are currently being used in a "live" project at NATS.

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

In the following discussion, we present some further details on the nature and characteristics of the case studies, and discuss their roles in the development and refinement of the information model and document templates.

3.1. Meeting scheduler

This case study was carried out as an application of the information model and document templates to a "greenfield" project. It is done based on the brief problem description as published in [4]. The case study is chosen because we are familiar with the application domain, so that we can concentrate on every aspect of the model/templates application.

First, we have identified the representative stakeholders, including meeting initiators, different meeting participants, meeting secretary, client representative, domain experts (convention consultants), representative of meeting venue, and system developers. Actual individuals are assigned to these roles through role-playing.

High level goals were identified by the stakeholders, based on the problem description and further considerations. The sketchy and imprecise goals or statements were refined and made clear by further discussion, and which was recorded through the templates' refinement structure. Conflicting requirements on dates, location, equipment, cost, and so on were prioritised according to the perceived values for relevant requirements. The goals were classified into appropriate categories as they were identified. We have found that it is useful to identify and consider individual aspects of the application during goal refinement. For the meeting scheduler, some of the aspects are meeting date and location, participants and location change, participants conflict resolution, participant interaction, and concurrent meetings. Goals identified for these aspects were arranged under their own sections or subsections in the requirements document.

During the requirements elicitation process, assumptions about the systems operating environment were surfaced, discussed and documented. For example, one particular assumption concerns the technology with which the system users access the system. A decision and assumption was made that users have access to the Web/Internet. Assumptions about natural law were also made clear, such as that people can not be at two place at the same time. The risks and authorities associated with the assumptions were identified and documented.

Acceptance criteria were defined by examining the system requirements and use scenarios. The client and the domain experts played a very important role in this process.

After the goals were sufficiently refined, a system architecture was developed, including components responsible for communication, information recording, conflict resolution, date search, schedule management, and system management. Services and QoS properties of the components were then identified and related to the goals.

In general, this case study has provided us with an opportunity to fine-tuning the information model, and valuable feedback to the structure and content of the templates. It has given us confident in the information model and document templates as to their effectiveness.

3.2. Flight data processing (FDP)

At the early stage of the project, we used the actual requirements document on FDP to motivate and validate the development of the information model. The preliminary model had been used to re-structure the requirements captured in the original FDP document. During this exercise, the issues and difficulties in using the preliminary model were analysed as the basis for refining the model and later designing the templates. For example, the distinction and relationship between goals and assumptions were made clear in this study with actual examples, and were proven to be beneficial in clarifying relevant system issues. (We note that there was no distinction between goals and assumptions in the original document). The study has found that it is necessary to have goal categories for proper management and guidance of goal identification, and to have organisational structures like sections to manage related goals. The broad distinctions among run-time system goals, design-time systems goals and process goals were also identified in this study.

The case study has also laid bear the lackings in the original requirements document (compared with the model). For example, the stakeholders were not identified in the document. As such, it was very difficult to find the appropriate person to clarify issues in the requirements. As mentioned above, assumptions were not made explicit, and the distinction between goals and assumptions were not easily identifiable, which often caused confusion. We should note that the original requirements document only contained very high level system requirements. As such there was no mention of system architectures, components, services or quality of service. This made clear the need to identify and discuss components in the context of system architectures.

In general, this case study has played a fundamental role in refining the information model and determining the content and structure of the document templates.

3.3. Short term conflict alert (STCA)

The STCA system was the reference application system for our study. It has been used in all stages of the study. In particular, both the HTML and DOORS implementations of the document templates have been fully populated with information from the STCA system. Again, this case study is

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

a retrospective application of the model, templates and tools to the information available in the documentation of an existing system. It had a reverse engineering flavour with the available documentation being detailed design documents and reverse engineered low level requirements documents.

In carrying out the case study, we first digested the available documentation and clarified outstanding issues with domain experts, to obtain an understanding of the domain and problem. Then we applied the model and templates to the system with tool support. This involved formulating, classifying and re-organising requirements. Certain requirements had to be extracted or formulated from the general descriptions and detailed design information found in the documentation. The requirements were classified according to the goal categories and organised into the sectioning structure of the templates.

Another task was to separate assumptions from goals. We went through the original requirements and design documentation in detail, and found that some requirements are actually assumptions about the system's operating environment and many assumptions are embedded in the surrounding discussions. For example, the assumption about the vertical separation at different flight levels was not mentioned at all in the original requirements document; rather, it was embedded in the discussions of the detailed design document. This is actually a very important assumption in view of a proposed change to the vertical separation standards.

In the original documentation, the system architecture was implied in the sense that the various processing components were *assumed* in the requirements description. These components are the coarse filter, three fine filters and the alert confirmation unit. The application of the document templates has given us an opportunity to discover and make explicit the implied system architecture. This also enabled us to identify the services of the components so that we can relate them to the goals and talk about the relationships between the components in precise terms.

To fully exercise the model and templates, we also made an effort to fill-in or simulate the information that is not in the original documentation, with the help of domain experts. Some of the information included stakeholders, values of goals, risks and authorities associated with assumptions, details of system architecture and interface, as well as the necessary relationships between the concepts. In particular, the refinement relationships between goals were introduced to provide rationale and traceability for the low level goals. We note that in the original requirements documentation, only the low level requirements were captured and documented. This is a crucial lacking because it hinders the understanding of the requirements without the high level goals and sponsoring stakeholders as their contexts. This has further emphasised the need and importance of retaining the high and intermediate level goals and maintaining the necessary refinement relationships between the goals.

During this case study, we have also briefly considered the need for requirements change. The actual case for change is referred to as Reduced Vertical Separation Minimum (RVSM). It changes the previous higher flight level criterion of 29,000ft to 41,000ft for aircraft that are equipped with RVSM. In effect, it reduces the vertical separation standards for RVSM-equipped aircraft from 2000ft to 1000ft, to increase the airspace's capability. This requirements change request violates the previous assumption about higher flight levels (i.e., above 29,000ft) as captured by the application of the information model. The traceability allowed by the information model has enabled us to quickly locate the system components that are dependent on the previous assumption and assess the architectural impact required to accommodate the change.

The system components affected by the above change request include the fine filters and alert confirmation. One possible architectural solution to the change request is to introduce a new component or service that hides the previous system parameter for the higher flight level criterion and delivers aircraft-specific higher flight level criterion. This solution minimises the change required to the system. This change case further highlights the benefit of capturing the requirements traceability and delineating the concepts involved (e.g., goals and assumptions) in the information model and document templates. In general, a requirements change will manifest into a change in the boundary between the system (defined by the goals) and the its operating environment (reflected by the assumptions).

4. Assessment

In this section, we reflect on the effectiveness of the information model, document templates and process and tool support for requirements engineering and architecture design, based on our experience with the case studies.

Completeness, clarity and structure of information capture. In existing practice, the requirements document is essentially a long list of requirement items with certain surrounding explanations. In contrast, the information captured through the concepts and relationships of the information model and document templates is much more complete with necessary clarity and structure. For example, the model and templates mandate the capture of stakeholders and the values they assign to goals. This information is particularly useful in providing traceability for requirements and resolving conflicts in requirements.

The clear distinction between goals and assumptions clarifies the boundary between the system and its operating environment, and assists the stakeholders and analysts in realising this distinction and removing much vagueness

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 **IEEE**

IEEE COMPUTER SOCIETY

in requirements. It also helps in managing requirements change and identifying the change's impact on the system. The explicit identification of risks associated with assumptions makes those relevant aware of the limits of the system and its potential for change.

The positioning of acceptance criteria in the model and templates further highlight their role in relating the system services and quality of service to the system goals. It enhances the traceability between the system architecture and the system requirements, which is not clearly provided in existing practice.

Traceability between concepts. In existing practice, there are hardly relationships between the different parts of the requirements document, which is partially due to the non-distinction of concepts. As such, it is very difficult to relate the various pieces of information and gain an overall understanding of the requirements. The relationships between the concepts in the information model and document templates set any of the requirements information in its network of understanding. This provides the necessary basis for better requirements understanding.

For example, the refinement relationship between goals serves as a mechanism for organising requirements, and provides the traceability between detailed requirements and high level requirements. As such, we can tell in which contexts a particular requirement is raised, and relate it to the relevant initial high level requirements. This is in clear contrast to the existing practice of primarily maintaining the low level requirements with their contexts being lost.

The system interface reflecting the assumptions materialises the boundary or relationship between the system and its operating environment. The explicit capture of such relationships supports the impact analysis of requirements change, such as the one outlined above for the STCA system.

The services and quality-of-service properties are the specific means that enable us to link the system architecture to the system requirements. This traceability again provides a tangible basis for analysing whether a system's design meets its requirements and how a change in requirements or design will affect the other side.

In general, the traceability allowed by the information model and document templates provides the concrete means for keeping system requirements and system architecture up to date and for actively utilising the requirements information in system analysis.

Architectural traceability. The explicit inclusion of the system component architecture and associated concepts in the information model and document templates is to reflect (1) that system architecture design is relevant to the elicitation and especially the refinement of system requirements, and (2) that the existing practice in requirements management rightfully includes the discussion of system architectures (to be able to document certain requirements). An example regarding the first aspect is that without the decisions on the system architecture in the STCA case study, the requirements on the coarse and fine filters and the alert confirmation unit can not be detailed. In general, system requirements as a whole can not be sufficiently refined to be operationalisable without the decisions made for and captured in the system architecture.

On the second aspect, the existing practice has already consciously or unconsciously pointed to the need to consider system architecture during requirements elicitation, although without explicitly addressing the relationships between system architecture and system requirements. We note that many of the decisions made during the operationalisation of goals in KAOS also affect the system architecture [2]. Making the system component architecture explicit and related it to the system goals and assumptions via services, QoS properties and system interface formally recognises the need to consider system architecture during requirements engineering, and provides the necessary traceability between system architecture and system requirements.

Clear structure and content for concepts and relationships. As mentioned above, existing practice does not dictate much structure for requirements capture, other than that the requirements are itemised, labelled and roughly categorised. The clear organisation and content structure definition for concepts and relationships in the document templates provide a better basis for documenting and managing requirements. Only through such clearly defined structures and content can system requirements be easily understandable and be *actually used* during the development and maintenance of a system.

Effective tool support. With the concepts, relationships and content definition structures of the information model and document templates, we are able to provide effective tool support that is not achievable in existing practice. For example, we are able to navigate around the requirements information according to the relationships of the model, which assists the documentation, understanding and evolution of the information. We are able to define focused views of the requirements information, which facilitates the inspection of the information from diverse viewpoints as required. The ability of being able to perform analytical queries on the information further strengthens its usability as questions like "Which requirements are not addressed by the system design?" can be answered. As mentioned earlier, it is possible to check the information's conformance to the requirements of given processes and standards through

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 **IEEE**

IEEE COMPUTER SOCIETY

straightforward enhancement of the tool. All the above is in clear contrast to using a basic word processor in managing the requirements information in the exiting practice.

Document templates as a guide to requirements engineering and architecture design. As intended by the suggested project start-up kit, the document templates together with the associated information model and tool support in fact provide a guide for requirements engineering activities. It sets out the different types of information that need to be elicited and documented, suggests the steps and techniques to be used in carrying out the requirements engineering and architecture design activities, shows the traceability that needs to be captured, and provides tool support for analysing the structural completeness and consistency of the requirements and architecture information. All these are not readily available in the existing practice, and contribute to the effective elicitation and management of system requirements and system architectures.

A standard for organisation-wide use. As intended, the availability of the document templates provides both a starting point and a standard for virtually all requirements engineering projects in an organisation. The organisation-wide application of the templates delivers value that is not possible before. It provides a standard way of maintaining system requirements and architecture information across the organisation in a uniform and consistent manner, and this information become a necessary part of the corporate memory. As such, it makes it actually practical to maintain and use such information with reasonable ease, and consequently alleviates the problem of corporate knowledge loss. At the same time, it facilitates organisation-wide learning, reuse and evolution of system information.

Management of requirements change. As discussed earlier, the information model and document templates facilitate the identification of the changes required relative to the existing goals and assumptions, and the scoping of their impact on other goals and assumptions and on the system architecture. This is due to the clear distinction and relationships of concepts in the information model and document templates. Such a capability is otherwise not possible in the existing practice.

5. Conclusions

In this paper, we have reported on our experience in designing and using the requirements and architecture management tool, TRAM. In particular, we have analysed its effectiveness relative to existing industrial practice in managing system requirements and system architectures. The tool has an underlying information model capturing the key concepts and relationships of requirements engineering and architecture design, embodies a set of document templates codifying the information model. and provides guidance to software practitioners in gathering and managing requirements and architecture information. All these components together form a project start-up kit for requirements and architecture management. The document templates have been applied to a number of real-life case studies with positive results. They are currently being used in a "live" industrial project at NATS. Further industrial applications of the templates and tool are also being explored.

The primary objective for TRAM is its practical usability. We plan to refine the tool based on our further experience with the industrial projects. We are currently carrying out an in-depth study of how to manage changes to system requirements and system architectures, and plan to incorporate the findings into TRAM.

Acknowledgement. The author would like to thank Anthony Brigginshaw, Wolfgang Emmerich, Anthony Finkelstein and Julia Sonander for their contribution and comments.

References

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addson-Wesley, Reading, MA, USA, 1998.

[2] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-Directed Requirements Acquisition. *Science of Computer Programming*, 20:3–50, 1993.

[3] W. Emmerich, A. Finkelstein, C. Montangero, S. Antonelli, S. Armitage, and R. Stevens. Managing Standards Compliance. *IEEE Transactions on Software Engineering*, 25(6):836–851, 1999.

[4] M. S. Feather, S. Fickas, A. Finkelstein, and A. v. Lamsweerde. Requirements and Specification Exemplars. *Automated Software Engineering*, 4(4):419–438, 1997.

[5] J. Han. A comprehensive interface definition framework for software components. In *Proceedings of the 1998 Asia-Pacific Software Engineering Conference*, pages 110–117, Taipei, Taiwan, December 1998. IEEE Computer Society.

[6] M. Jackson. The World and the Machine. In *Proceedings of the 17th International Conference on Software Engineering*, pages 283–292. ACM Press, 1995.

[7] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In *Proceedings of the 5th European Software Engineering Conference*, pages 137–153, Barcelona, Spain, September 1995. Springer.

[8] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, Upper Saddle River, NJ, USA, 1996.

[9] W. Smith. *Best Practices: Application of DOORS to System Integration*. QSS Quality Systems and Software, 1999 So. Bascom Ave., Suite 700, Cambell, CA 950008, 1998.

1080-241X/03 $17.00 © 2003 **IEEE**

IEEE COMPUTER SOCIETY