TRAM: A Tool for Requirements and Architecture Management

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

Abstract

Management of system requirements and system architectures is part of any software engineering project. But it is usually very tedious and error prone. In particular, managing the traceability between system requirements and system architectures is critical but difficult. In this paper, we introduce a tool, TRAM, for managing system requirements, system architectures and more importantly the traceability between them. Its primary design objective is "being practical" and ready for practitioners to use without much overhead. The issues discussed in this paper include an information model that underlies the capture of requirements, architectures and their traceability, a set of document templates implementing the information model, and the support tool.

Keywords: *Requirements management, system architectures, software engineering tools.*

1. Introduction

Continued management of system requirements, system architectures and the traceability between them provides critical support for 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.

In this paper, we introduce 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.

The paper is organised as follows. We first review the design objectives for the tool and document templates. Next, we introduce the information model for capturing requirements, architectures and their traceability. We then discuss the templates design and tool implementation, before finally concluding this paper. Three case studies of applying the document templates and tool to real-life systems, and a comparative assessment of the tool relative to existing practice are reported in a separate paper [5].

2. Design objectives

The set of document templates for requirements and architecture management provide a tangible basis for carrying out requirements engineering and architecture design activities. They serve not only as a starting point and scheme to organise and manage

60

system requirements and architectures, but also provide guidance and serve as a check list for the relevant activities. The following sets out the specific objectives of the document templates and tool support.

- Requirements management for projects and organisation: Requirement management should not only be a project-oriented issue, but also an organisation-wide issue. There should be consistency, consultancy and systematic knowledge sharing and leverage across projects within the organisation. The document templates should serve both project-level and organisation-level purposes.

- Information and process of requirements engineering: Both the information and process aspects are critical to requirements management. The document templates should capture the basic requirements information and the necessary information facilitating the requirements engineering process, and provide process guidance for the capture of the requirements information.

- Requirements capture, change and evolution: The document templates should not only be about the capture or representation of system requirements. They should also accommodate and facilitate requirements change and evolution.

- Traceability between system requirements and system architecture: The traceability between system requirements and system architecture is of critical importance. It will help to answer questions like: "Is requirement A being addressed by the architecture design?" and "Which system components are relevant to meeting requirement B?" Similarly, such traceability will be invaluable in assessing the impact of a proposed requirements change. As such, the document templates should also facilitate and capture the system architecture and the traceability between requirements and architecture.

- Practical, immediate and incremental uptake: The document templates should allow immediate *ready* uptake in practice with minimal training. This requires that the templates should closely related to existing practice. On the other hand, they should also allow incremental uptake of advanced features for further improved requirements and architecture management. We emphasise that this is the primary objective for the design of the document templates and the tool support.

The tool support for requirements and architecture management to be introduced in the following sec-

tions has been designed with the above objectives in mind. It involves three major aspects:

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.

This project start-up kit has been applied to a number of case systems during and after their development for the purpose of refinement and validation [5]. They are currently used in a "live" industrial project at National Air Traffic Services (NATS).

3. An information model

Following the design objectives, 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 [4].

61

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.

3.1. The model

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 practice, and are further refined through application to real-world industrial projects. Figure 1 presents the information model in an entity-relationship diagram. 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 confidence. 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.

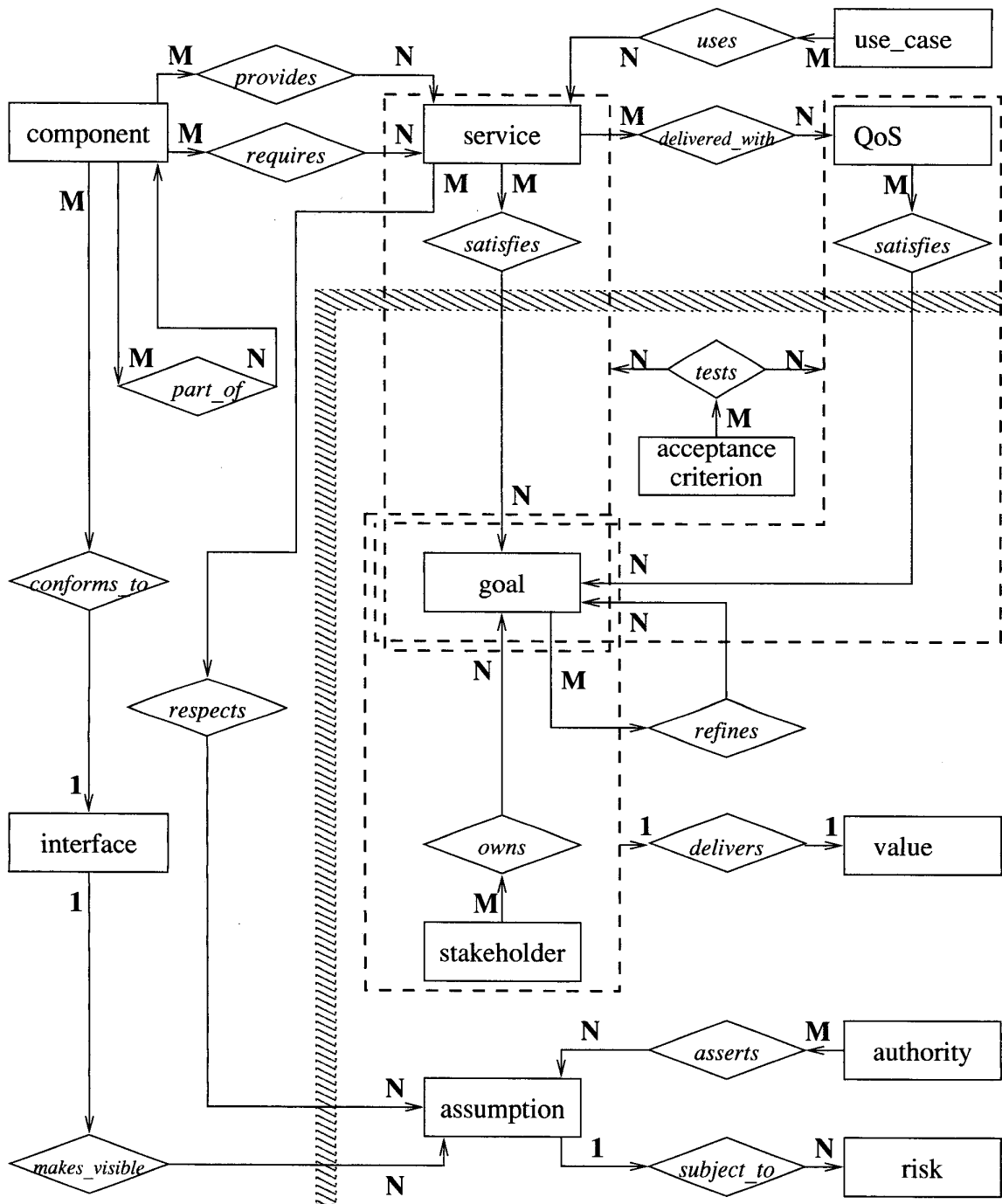In general, the acceptance criteria should be

62

Figure 1. Core information model for requirements and architecture management

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).

3.2. Examples

In this section, we illustrate the core information model with examples drawn from an air traffic Short Term Conflict Alert (STCA) system currently in operation. The main functionality of the STCA system is to test periodically the state of all aircraft/system tracks under its control to determine whether any pairs of tracks fulfill the conditions required to declare a conflict alert. Any declared alert will be displayed to the air traffic controller. Essentially, a conflict alert for a pair of aircraft indicates that the two aircraft are too close to each other. The STCA system interacts with other Air Traffic Control (ATC) systems, including Multi-Radar Processing, Control and Monitoring, Workstation Display Management, Recording and Display, and Support Information Database.

Stakeholders: For the STCA system, some of the stakeholders are:

- *Air traffic controllers:* The air traffic controllers are the primary users of the system.

- *Pilots:* The STCA system directly concerns the pilots of the aircraft in the system tracks tested.

- *Civil Aviation Authority:* This is the the national statutory body for air safety.

- *STCA Design Authority:* This is the organisation responsible for system development and implementation.

Goals: The stakeholders of the STCA system state various goals. Some high-level goals of the air traffic controllers include:

- *g1:* All short term conflicts are declared in time.

- *g2:* Newly identified alerts must be displayed for a minimum period of time (MINDISPLAY-TIME) for controllers to locate the aircraft concerned.

The pilots may state that

- *g5:* Relevant pilots are notified of conflict alerts (through air traffic control) *in time* for taking collision avoidance actions.

As an example of goal refinement, goal *g1* can be refined into the following lower level goals:

- *g11:* Identify all pairings of tracks that are of *potential concern* (according to set criteria) for conflict alert. This is to quickly reduce the track space for further complicated processing.

- *g12:* Eliminate, from the set of potentially conflicting track pairs, those pairs that do not satisfy the conflict alert condition.

- *g13:* The accumulative processing time for the tasks involved should be within the allowed maximal delay.

Values: The air traffic controllers may give the following values to their initial goals:

- *g1's value:* Achieving *g1* will greatly help the air traffic controllers in identifying all conflict alerts and taking appropriate action to avoid any collision.

- *g2's value:* Achieving *g2* will help the air traffic controllers to identify all alerts before they disappear.

Acceptance criteria: The following are some acceptance test cases (i.e., more concrete or refined acceptance criteria) aimed at the identified goals.

- *ac1:* The system should have a successful conflict alert detection rate of 99.99% and all severity 1 alerts are reported. This should be established from a set of at least 100,000 sample system tracks in normal daily operation situations.

- *ac2:* For the given set of devised "collision/near-collision" cases, the system should *always* identify them as having severity 1 status and declare alert. (Note: The devised cases should be given.)

Generally speaking, the more refined the goals are, the more concrete the acceptance test cases can be. We note that acceptance criteria usually relate to specific use scenarios.

64

Authorities: For the STCA system, the authorities may include:

- *Avionics experts:* These experts provide domain expertise, and state natural limits that the STCA system operates in. Such knowledge is particularly important in refining high level goals into concrete goals.

- *Radar data processing representatives:* These representatives state limits/facts about the radar data that the STCA system relies on.

- *Civil Aviation Authority:* In this role, the Authority states relevant laws and regulations.

Assumptions: The Civil Aviation Authority and the avionics experts state that

- *a1:* At higher flight levels, presently above 29,000ft, the vertical separation standard is increased from 1000ft to 2000ft.

The radar data processing representatives state that

- *a2:* The aircraft track information provided to the STCA system by radar data processing functions includes lateral and vertical tracking.

Risks: A risk related to assumption *a2* is that

- Due to malfunction of radar equipment and/or the radar data processing facility, certain track information may be inaccurate or missing.

Interface: Two interface conditions that refine assumption *a1* are

- *i1:* The flight levels of the system tracks are available to the system as input.

- *i2:* The high flight level criterion (presently 29,000ft), and vertical separation standards for higher and lower flight levels are available to the system as (environment) control parameters.

Components: The STCA system has the following major system components:

1. Coarse filter: This component identifies the potentially conflicting track pairs.

2. Fine filters: There are three fine filters, namely, linear prediction filter, current proximity filter, and manoeuvre hazard filter. These filters process in parallel the potentially conflicting pairs produced by the coarse filter, and produce filtered track pairs.

3. Alert confirmation: This component takes the results from the three fine filters (i.e., the filtered conflict track pairs), eliminates the unnecessary alerts, and generates alert messages.

Services: Services are capabilities of the system that are devised to satisfy the goals in the sense that they operationalise the goals. In general, therefore, we need to sufficiently refine the goals to identify services. One service of the STCA system is

- *s1:* Confirm and generate conflict alerts by applying confirmation logic to the filtered conflict pairs.

This service contributes to the satisfaction of goal *g12*, and is provided by the *alert confirmation* component in the system architecture.

QoS: Service *s1* has the following quality property:

- *QoS1:* Radar data are processed at real-time rate, and conflict alerts are generated within the given maximal delay (MAXDELAY). This is a *performance* property.

This quality property satisfies goal *g13*.

Use cases: An example use case is as follows:

- *uc1:* Two aircraft enters the controlled airspace, and gets too close to each other (vertically and laterally) so that the separation requirements are breached. Consequently, a conflict alert is identified and reported by the STCA system. The alert is maintained for a minimum number of cycles, even if the two aircraft divert from each other immediately after the separation breach.

This use case exercises the service *s1* identified above and a range of other services provided by the system.

4. Templates design and tool implementation

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, and selecting and accommodating the

65

targeted tool support. We have designed and 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.

In this section, we first discuss the content design and refinement for the templates. Then we examine the specific features of the templates in the two forms of implementation.

4.1. Templates content design and refinement

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. These additional features are the focus of the following discussion.

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 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. Again, higher level goals tend to concern many categories while lower level goals can be allocated into single categories. In general, it is beneficial to be aware of the broad and finer categories that a goal belongs to. In particular, trying to allocate categories to an identified goal will force the stakeholder and the analyst to think carefully about the goal's role and place in the system. 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 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 alongside the assumption statement. 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. 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. Although the system architectures, inter-component interactions and architectural constraints are not explicitly shown in the information model, 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 an informal, semi-formal or formal notation.

The services are in general operationalised system goals. They are assigned a service category. The service categories are system specific and usually re-

66

flect 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.

The document templates also capture some additional information. This includes information for project identification and management, the definition of domain concepts (i.e., a glossary), and an optional domain model. This information forms an important part of the corporate memory.

4.2. Tool 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 the *System Architecture Document*. The division is shown in Figure 1 by the filled thick line. 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 2 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. The ability of being able to perform such queries is immensely useful in requirements and architecture management.

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].

5. Conclusions

In this paper, we have introduced a tool for requirements and architecture management, TRAM. It involves an underlying information model capturing the key concepts and relationships of requirements
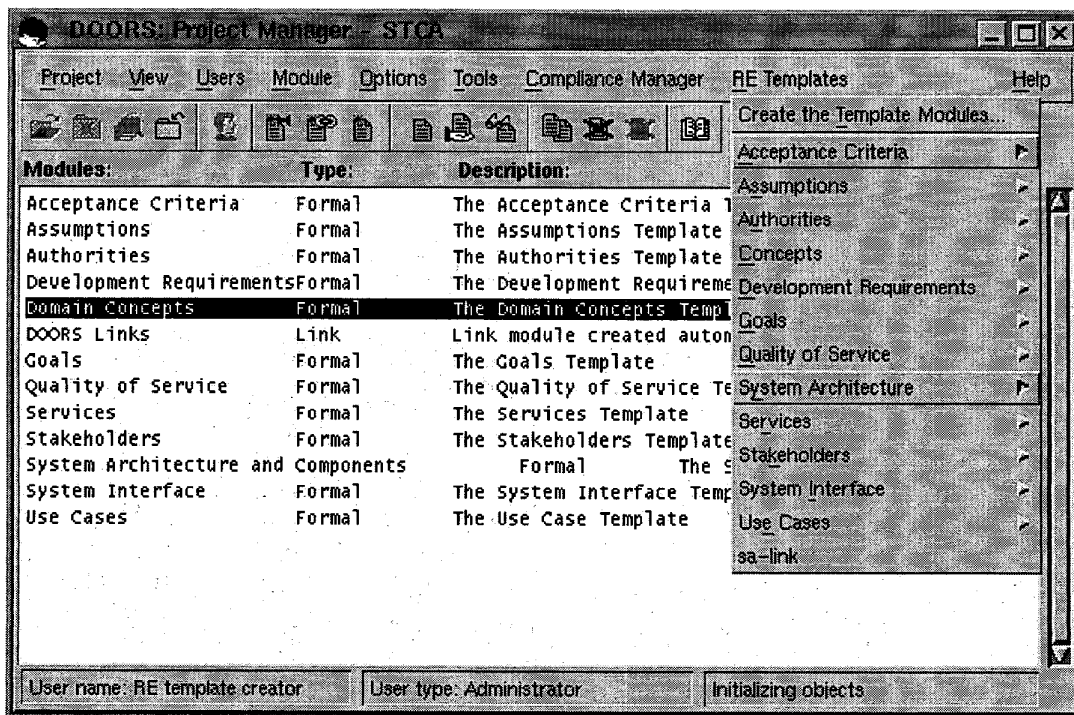
67

Figure 2. DOORS implementation of document templates

engineering and architecture design, a set of document templates codifying the information model, and the actual tool implementing the templates and providing guidance to software practitioners. 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 [5]. 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 further refine the tool based on our experience with the industrial projects. We are currently carrying out an in-depth study of managing changes to system requirements and architectures, and plan to incorporate the findings into TRAM.

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] 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.

[5] J. Han. Experience with designing a requirements and architecture management tool. In *Proceedings of the 2000 International Conference on Software Methods and Tools*, 10 pages (to appear), Wollongong, Australia, November 2000. 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, USA, 1998.