Developing Requirements in an Established Domain Using Tags and Metadata

Leyna C. Cotran
Department of Informatics
University of California, Irvine
Irvine, CA 92697-3440 U.S.A.

lcotran@uci.edu

Richard N. Taylor
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 U.S.A.

taylor@ics.uci.edu

Abstract— **Requirements for new aerospace applications, such as missile or satellite control systems, draw heavily from similar, previously built systems. Both requirements and varied architectural artifacts from prior systems may be consulted. The application domain is quite complex, so "reuse" brings clear advantages. Unfortunately numerous problems are apparent too. For example, while prior artifacts are potentially rich in information, if they are of low quality they may hinder development of high quality requirements for new systems. The Co-Evolvable Traceable Requirements and Architecture Network (COTRAN) technique presented here addresses the challenge of productively using information from prior systems while targeting development of properly scoped, testable requirements. The technique centers on iteratively building a requirements matrix characterizing the application and tagging artifacts based upon the metadata found in the matrix. This enables information to be correlated across artifacts and used consistently. Using the requirements matrix, contextual information about the system provides the basis for capturing clear requirements.**

Keywords- Requirements/Specifications elicitation methods, methodologies, software architecture, aerospace systems, tags, metadata

I. INTRODUCTION

The engineering of software requirements for aerospace applications is arguably in need of significant improvement. Aerospace systems are highly complex, being comprised of multiple also-complex subsystems, for which software is a key integrating technology. Numerous companies are involved in the creation of most spacecraft systems; requirements specifications form the basis for the contractual relationships among the organizations.

Unfortunately, despite the importance of good requirements [5], typical requirements engineering practice in this domain is beset with a variety of problems. For example, requirements specifications all-too-frequently arise that are non-testable, ambiguous, or not properly scoped. Critical contextual information is also frequently lost. Such information has the potential for providing insight regarding the motivation for inclusion of a requirement, guidance for implementation choices, or indication of one requirement's relative priority over another, should the need for requirements triage occur.

This is ironic, since spacecraft systems are not new: aerospace companies have been building these systems for decades and there is substantial experience upon which to draw. Requirements engineering professionals have long worked in the domain and the value of good requirements specifications is widely acknowledged. Many major aerospace programs are upgrades to previous programs, implying that the specifications for the upgraded system will be similar to those of the previous system. At a minimum there will be commonality with respect to the specific application domain and most likely there will be significant overlap in the constituent subsystems. While this paradigm is potentially effective, the reality is that these previous systems likely include poor requirements, lacking the information needed for understanding and capturing new, high-quality requirements.

The practice that provides the way out of this thicket is referencing architectural artifacts from the pre-existing systems as well as those that may be emerging concurrently for the new system. That is, a key aspect of the oft-missing context is the systems and (partial) software architecture within which the new system/application is to be conceptualized. These artifacts correlate in various ways to prior requirements and, potentially, to the new requirements under development. They provide significant information to help better understand prior requirements and develop new ones.

We view the problems with requirements engineering for spacecraft systems as open for improvement, and that improvement is indeed possible through the leveraging of developer experience, expertise, and the architectural artifacts available. This paper outlines a new requirements engineering technique created on this foundation. The Co-Evolving Traceable Requirements and Architecture Network (COTRAN) technique incorporates two primary innovations. First, COTRAN uses a tagging approach to connect requirements to correlating architecture and design artifacts, including artifacts from prior systems. Second, COTRAN uses a matrix approach to index and organize requirements by correlating architecture information found in these artifacts. The matrix approach works particularly well in aerospace because a large wealth of domain knowledge can be used to develop critical requirements.

The remainder of this paper is organized as follows: Section 2 describes metadata tags usage; Section 3 discusses

978-1-4577-0941-8/11/$26.00 ©2011 IEEE

how to build the requirements matrix with the COTRAN technique; Section 4 provides an example; Section 5 discusses writing requirements using the matrix. We conclude and describe our future work in Section 6.

II. USE OF METADATA TAGS

Metadata is "a set of data that describes and gives information about other data [4]" One common popular usage of metadata is with digital photo management [4]. Digital images from modern cameras include metadata about the photo: location, exposure, and date, for example. This metadata can serve as keywords for searching or indexing on social digital sites such as Flickr. Names of people in these photos are metadata that is tagged by users (rather than the camera).

Similarily, tagging metadata found in architecture artifacts to support requirements development provides two benefits. One, these tags provide an index to the topical area of these artifacts. Two, they provide a mechanism for organizing this data.

Architecture artifacts are rich in requirements-relevant data, and establishing tags for this data allows us to correlate this architecture-based information to requirements. For example, architecture artifacts contain data that describes behavior, interfaces, and functionality of the system. These types of information are examples of the type of metadata that provide a basis for understanding requirements.

Architecture artifacts are linked through metadata tags, providing associations between the information found in these artifacts and meaningful requirements. By organizing requirements around a metadata tags structure, we gain the capability to correlate specific software requirements to specific aspects of the architecture.

III. COTRAN TECHNIQUE: BUILDING THE REQUIREMENTS MATRIX

The COTRAN requirements engineering technique builds a matrix for requirements through correlating critical information found in software architecture artifacts. We incorporate two primary innovations. First, we use a tagging approach to connect requirements to architecture artifacts from either prior systems, or the system-to-be. Second, we use a matrix to organize the data found in these artifacts. This further improves the quality of requirements by making it easier to assess whether a requirement is appropriately scoped and testable.

We believe that a sound approach to requirements synthesis involves a significant intertwinement with architecture development. To execute this paradigm, requirements have to be developed in parallel with architecture artifacts.

To build the matrix, we must establish software architecture artifacts that define behavioral and functional data of the system. This data is tagged with a meaningful keyword. The tagged metadata content is captured in the matrix, where this content is indexed by their tags. Once this matrix is built, we can assess the correlated information for use in meaningful requirements.

A. Establish Software Architecture Artifacts

Architecture artifacts include a *data dictionary*, *context diagrams*, an *operational concept*, the *software architecture*, and *state diagrams*. Such artifacts frequently exhibit both requirements choices and design decisions (i.e. software architecture, "the set of principal design decisions made about the system" [1]). Architecture artifacts evolve during the development of a system, meaning that requirements must continuously adapt with these evolutions. Together these artifacts imply a network of information that can provide various views of the system.

The *data dictionary* establishes terminology, aiding enforcement of consistency of metadata tags to be used across artifacts. *Context diagrams* provide information on the system's actors and how these actors interplay. This provides scope for the requirements problem. An *operational concept* describes how people, systems, and all the elements of the mission architecture will interact to satisfy the mission goals. *Software architecture* encompasses design decisions such as structure, functionality, and interdependencies. *States* of the software system capture the transitions, triggers, and events of the system.

B. Define the Metadata

After the architecture artifacts have been established, we examine these artifacts for metadata – that is, data that will augment our understanding of requirements. The criterion for metadata is based on the software architecture artifact and understanding what information that artifact provides.

For example, data in *state diagrams* describe the systems' behavioral transitions from start to finish. By tagging aspects of these behaviors in a state diagram, we can correlate the requirements that deal with system behaviors. States such as *operate* or *idle* will have a number of associated requirements that deal with those particular states. The metadata itself might also be the behaviors *operate* or *idle* in the state diagram, but now the information that deal with these two particular states are tagged for correlating potential *operate* or *idle* requirements. Important metadata found in *context diagrams*, as another example, are the definitions of interfaces and key players that context diagrams inherently capture. Metadata found in the system's *operational concept* includes objectives, goals, and planned development paths. This information that each of these unique artifacts provide will shape potential requirements [6].

The *software architecture* artifact is also very rich in metadata for requirements. This critical metadata includes architecture styles and data flow between architecture components. An architecture style, for example, reflects a specific category of systems, therefore providing a significant amount of context about what the system can do and how it will do it. In a product line context architectural styles may also be accompanied by a set of reference requirements. In particular, domain specific architectures [1] exist in the aerospace domain and knowledge about these domains is embedded in these specific architectures. This provides significant data for correlating requirements.

C. Tag Metadata in Architecture Artifacts

Once metadata in the software architecture artifacts is determined, we then *tag* this data. Metadata **tags** in the COTRAN technique are annotated keywords placed on the data content in the software architecture artifacts. Using the tag definitions in the data dictionary, we tag relevant elements of the architecture artifacts that comply with their respective definitions captured in the data dictionary.

Tags are intended to be repeated within the same artifact and cross cut through other artifacts (it is expected that there would be a number of tags related to *telemetry* or *interface*, as examples). For distinct tags of the same type, it is necessary to add an identifier to these tags (i.e. *telemetry 01*, *telemetry 02*) to distinguish the tagged content. Should the same piece of metadata reappear across artifacts (i.e. data01 appears in multiple artifacts), this means that this piece of metadata is captured at a different granularity of detail. In this case, *telemetry 01* would appear several times in the requirements matrix. It is up to the engineer to reconcile the meanings of all the *telemetry* tagged content.

D. Create the Requirements Matrix

We define a requirements matrix as a table structure that captures the information tagged in the software architecture artifacts. It is intended that this information placed in the matrix will allow the engineer to further decompose the information to requirements. We define each row in the matrix as a *tuple* for one requirement. This means that each tuple defines the associated architecture artifact data per tagged item. It is also expected that the engineer will further expand the matrix with additional attributes: specifically the *requirement* itself, the requirement *rationale*, and the requirement *verification success criteria*. Table I shows the matrix attributes.

The matrix provides an organization mechanism for assessing requirements in *groups* or *individually*. Looking at our brief earlier *telemetry* metadata tag example, we would have several *telemetry* tags in the matrix. These *telemetry* tags would be organized together in the matrix, where *telemetry01* till *telemetry0n* will be grouped together. Therefore, the engineer can assess all the *data* tuples together as a group, or investigate one individual tuple with the *telemetry* tag.

TABLE I. REQUIREMENTS MATRIX ATTRIBUTES

Matrix Attribute	Definition
Metadata Tag	Name of Metadata Tag
Metadata Tag ID	ID assigned to metadata tags
Architecture Artifact	Artifact where the metadata was tagged from
Description of Tagged Content	Description of the content tagged with a particular metadata tag
Requirement	Requirement captured from correlating metadata content
Rationale	Rationale of requirement (using architecture metadata as the basis)
Verification Success Criteria	Specific verification criteria that demonstrates the requirement was fully met

Using metadata found in relevant architecture artifacts forces the requirements matrix to be focused around architecture. The tags can be filtered within the matrix for stakeholders that are focused on a specific area of the architecture.

In our current implementation of the COTRAN technique, we built these matrices in the DOORS environment. We build our architecture artifacts using commercially available modeling tools, and embed the tags in artifacts. The tags are assigned identification numbers and we import these tags and tag IDs into DOORS modules. We then incorporate our decomposition of the metadata into requirement statements, with the correlating metadata of architecture information, rationale, and verification success criteria.

IV. FIRESAT EXAMPLE

The COTRAN technique can be illustrated using the academic FireSat spacecraft example [7] to demonstrate a simple behavior transition requirement. The objective of FireSat is to orbit the earth and detect forest fires. Once detected, these forest fires are reported to the FireSat ground station. In this example, we want to express the requirement of specific time transition from the FireSat's idle state to operate state. Timing requirements are found, for example, in a concept of operations document (dealing with orbit phases), an architecture dataflow (defining clock timing interfaces), and a states diagram (capturing the state transitions between different behavior phases and timing between phases).

In this example, we see that the timing need for FireSat's Idle to Operate is at 5 minutes. Note, that this timing need emerges over different artifacts. Each artifact provides a different perspective of this requirement, and so we tag this data appropriately with *timing01* throughout the artifacts that capture this 5 minute transition. The *timing01* tag is now embedded in these architecture files. What is demonstrated here is a requirement that has emerged over several tightly associated architecture artifacts.

As we develop these artifacts, we start building our matrix. Since we have now established this timing need among artifacts, then we add it as a requirement in the matrix. *timing01* is an index to the timing requirement of Idle to Operate in 5 minutes. Should the system evolve (perhaps that timing requirement will change to 7 or 8 minutes), then we locate the requirement in our matrix per the *timing01* tag, which is now mapped to all relevant artifacts and information.

V. WRITING THE REQUIREMENTS

Once the matrix is established, the engineer is then to add the *requirement* derived from the tagged information, the *requirement rationale*, and the *verification success criteria* based on the metadata tags and content that is tagged. The engineer needs to use their expertise and knowledge to write

the requirement statement. The matrix provides the contextual information to do this. The requirement statements focus on the architecture content. The tags provide the organization mechanism for these requirements (i.e. *data* requirements, *telemetry* requirements). Using the data in the matrix, a requirements specification is created by extracting the requirement, its correlating architecture correlation, the rationale, and the verification success criteria. With this contextual metadata, the written requirement statement has more data in order to interpret the requirement.

Traditional aerospace industry practice is to capture requirements in a document with simple "shall" statements using various military standards for writing requirements. We believe that an architecture rationale will give more merit as to why a requirement exists. Using our matrix approach, the requirements data traces to tagged architecture content that provides that rationale basis.

Verifiability is another critical factor to requirements development. Capturing verification aspects of requirements helps promote testing approaches earlier in the development cycle. The COTRAN technique enforces the capturing of the verification success criteria of the requirement in the requirements matrix; that is, capturing the steps or resultant measurements that will demonstrate that the requirement has been successfully proven.

Requirement statements alone do not provide enough information or context to understand the meaning of the requirement or the consequence of executing the requirement. The matrix provides the mechanism to correlate architecture metadata information together so that writing the requirement has context and meaning.

VI. FUTURE WORK

The COTRAN technique has been applied in two requirements engineering projects at a large government-contracted aerospace company. One study is complete and currently under analysis; the other is on-going. Both studies involve "live projects", rather than artificial exercises.

Our future direction with the COTRAN technique is to further develop the matrix approach into an ontology-based [2] approach. An ontology is appropriate for aerospace-based systems as these systems are well defined in particular domains (e.g., spacecraft, satellites) where concepts and entities are defined and are strongly correlated to the development of a domain-based system.

There are a number of benefits with an ontology approach. First, ontologies are an accepted way of organizing domain information [2]. Because ontologies are a representation of a set of concepts in a domain and the relationships between those concepts [2], it is reasonable to consider ontologies for aerospace domain-based products.

Second, ontologies provide a framework for organizing critical data about a system. Ontologies capture common elements of a system through attributes and relationships from domain concepts.

Third, there is flexibility in such ontologies. Ontologies can capture a number of different elements and can be tailored to project-specific needs. Therefore, tracking dependencies to requirements information is facilitated. The contextual data found in architecture, design approaches, or verification aspects can be tracked – the ontology can be molded to fit the needs of the project's requirements development.

The current matrix approach captures the dependencies of requirements information to architecture, design decisions, and testability criteria. The COTRAN technique's current use of a matrix to correlate the metadata found in architecture artifacts has significant potential in resulting to an ontology-based approach.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under grants CCF-0917129 and CCF-0820222.

REFERENCES

[1] Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy. Software Architectures: Foundations, Theory, and Practice. John Wiley & Sons, 2008.

[2] Gruber, T. R. 1993. A translation approach to protable ontology specifications. Knowl. Acquis. 5, 2 (Jun. 1993), 199-220.

[3] Ames, M. and Naaman, M. 2007. Why we tag: motivations for annotation in mobile and online media. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (San Jose, California, USA, April 28 - May 03, 2007). CHI '07. ACM, New York, NY, 971-980.

[4] Sarvas, R., Herrarte, E., Wilhelm, A., and Davis, M. 2004. Metadata creation system for mobile images. In Proceedings of the 2nd international Conference on Mobile Systems, Applications, and Services (Boston, MA, USA, June 06 - 09, 2004). MobiSys '04. ACM, New York, NY, 36-48.

[5] Nuseibeh, B. and Easterbrook, S. 2000. Requirements engineering: a roadmap. In Proceedings of the Conference on the Future of Software Engineering (Limerick, Ireland, June 04 - 11, 2000). ICSE '00. ACM, New York, NY, 35-46.

[6] Axel Van Lamsweerde. Requirements Engineering: From System Goals to UML models to software specifications. John Wiley & Sons, Ltd. UK. 2009.

[7] Jerry J. Sellers. Understanding Space: An Introduction to Astrodynamics, 2nd Edition. McGraw-Hill Companies, Inc. Boston, MA. 2004.