Goal-Driven Software Reuse in the IV&V of System of Systems[*]

Thomas W. Otani, James Bret Michael, Man-Tak Shing
Naval Postgraduate School
Monterey, California, U.S.A.
{twotani, bmichael, shing}@nps.edu

Abstract

This paper addresses the need to increase the effectiveness and productivity of independent verification and validation (IV&V) of complex system-of-systems software via software reuse. It builds upon our previous work on reusing the system reference model (SRM) artifacts in the IV&V of system-of-systems software and presents a framework for organizing the reusable artifacts according to a common set of business goals. We demonstrate the proposed framework using NASA science missions as examples.

Keywords: Software reuse, Goal-driven reuse, IV&V, System of systems, space systems

1. Introduction

In [1] Caffall and Michael describe the prevalence of systems of systems used by the National Aeronautics and Space Administration (NASA) and its partners such as the European and Japanese space agencies. There is a long history within the space systems community of reuse-in-the-large: creating systems of systems from legacy systems and new developments to support new missions. For example, one would like to leverage existing space vehicles and communication systems to support new space-exploration missions.

Although the reused systems may have a long history of service and have undergone extensive verification and validation (V&V), when they are reused as part of system of systems their behavior needs to be reverified and revalidated for the system-of-systems context in which they will operate in support a mission. *Validation* refers to ensuring the correct product is built (i.e., all the requirements for the product are identified and correctly specified), whereas *verification* refers to ensuring the product is built correctly (i.e., all the stated requirements are satisfied by the product). *Independent validation and verification* (IV&V) means that a team independent from the developers conducts the V&V of the system.

We were tasked by the NASA IV&V Facility to develop a framework for conducting computer-aided formal V&V. As we developed that framework it became obvious to us that there is an opportunity to make such a framework more palatable to its users by providing them with a means to leverage V&V artifacts produced from past IV&V of systems and reuse them in newly formed systems of systems or plugged into existing systems of systems. In this paper we introduce a goal-driven approach to reuse of assurance-related artifacts to support the conducting IV&V on systems of systems. The approach is general enough to apply to the evolution of single systems too.

In [2], we discussed why the traditional, mainly manual, IV&V methodology is inadequate and proposed a new software automation framework for computer-aided formal V&V. One of the key contributions of our framework is the concept of an executable system reference model (SRM) that utilizes lightweight formal methods. The SRM captures the system behavior precisely by identifying: (a) what the system should do, (b) what the system should not do, and (c) how the system should respond under adverse conditions. The adoption of SRM-supported V&V is a step in the right direction, but it not does not guarantee that an IV&V effort will be successful. Just as programming in a high-level language is much more efficient and effective than programming in assembly language or microcode, the use of the SRM can improve the IV&V team's effectiveness and efficiency

[*] The research reported in this article was funded in part by a grant from the National Aeronautics and Space Administration. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotations thereon.

U.S. Government work not protected by U.S. copyright
DOI 10.1109/SOSE.2010.464

at detecting and resolving problems at the system-requirements level without getting sidetracked by lower level system artifacts: This is in concert with the software testing adage that one should conduct testing with a model at the right level of abstraction to answer questions about the behavior of the system at that level (e.g., do not try to find requirements-level failures by conducting module-level testing). One of most effective ways to increase the productivity of programming in a high-level language is to adopt a reusable code library such as the Java *Application Programming Interface* (API). We would like to achieve the same end when using a SRM by building a library of reusable analysis and design artifacts. In [3, 4], we presented our initial foray into this area. In this paper, we discuss our current effort to expanding the reusable artifacts to include business goals.

The paper is organized as follows. In Section 2, we describe the SRM and discuss our previous efforts at reusing SRM artifacts. Then, in Section 3, we explain our work on improving reusability by including business goals as the guide for reuse. We propose to classify the business goals into three categories, and based on this categorization, we present a framework for a reuse library that will allow the modelers to search and reuse assets effectively in Section 4. Section 5 summarizes the key accomplishments.

2. SRM and Reuse

Much of this section is a summary of our previous research on reuse reported in [3, 4]. We will describe the SRM and discuss how the SRM artifacts can and should be reused.

An SRM is composed of Unified Modeling Language (UML) artifacts, such as use cases, activity diagrams, sequence diagrams, and class diagrams, along with formal executable assertions for specifying system behavior. In the SRM, formal assertions are enhanced statechart diagrams [5], with each system requirement represented by a single assertion. The IV&V team uses the assertions to instrument the developer's software. When an assertion pops during the software testing (via automatic generation and running of test cases based on the SRM, the details of which can be found in [2]), this is an indication that either one of two things has occurred: the software has failed to meet a requirement or the assertion itself is not a correct representation of the real requirement (i.e., the cognitive understanding of what the stakeholder expects in terms of system behavior); this is why we refer to this as computer-aided formal validation.

As is the case for any formal modeling and analysis tool, it takes a nontrivial level of effort by a novice to become proficient at using the SRM approach; use of formal methods, let alone cutting-edge techniques like execution-based model checking, are not part of the knowledge base or skill set of most software or systems engineers. Even for a well-seasoned practitioner, creating and maintaining an SRM is not a simple task. To aid both the novice and expert practitioners working with an SRM, we advocate the adoption of a reuse library. An effective reuse library can improve the quality of the SRM and reduce the time and cost involved in developing the model.

We proposed a framework for a reuse library in [4] and described a way of reusing libraries of statechart assertions in [3]. Instead of defining assertions from scratch, the modelers reuse assertions from a library. Correctly constructing assertions is one of the more difficult tasks in the development of an SRM, so by reusing previously tested assertions from a library the team conducting assurance improves its chances of starting out with the right model against which to judge the behavior of the system under review.

Our initial efforts were focused on reusing individual artifacts, such as assertions or activity diagrams. There are two types of reuse in the SRM approach: *adoption reuse* and *instantiation reuse*. Adoption reuse involves copying and using existing artifacts in another product. An artifact can be adopted as is or with modifications. In contrast, instantiation reuse entails creating a concrete artifact from a generic template. In [3] we described an instantiation reuse of the assertions library. The modelers create concrete assertions and test scenarios from the assertion patterns and validation test-scenario patterns in the reuse library. In [4], we described a framework that supports both types of reuse.

The ultimate goal of our research, as mentioned in [3], is the reuse of *assets*. An asset is a collection of related artifacts. For example, instead of finding and reusing individual artifacts, the modelers will locate and reuse a collection of artifacts associated with a single high-level task such as "put the spacecraft into an orbit." We describe in this paper a step toward achieving this goal.

3. Reusing Business Goals

There are many ways to classify space missions. Possible mission classifiers include but are not limited to: mission types (e.g., Orbiter vs. Lander), location/ destination (e.g., Earth Orbit vs. Non-Earth Planetary Orbit), orbit type (e.g. Low Earth Orbit, Polar, or Geocentric), number of orbiters (e.g., Solo vs. Cluster), mode of operations (e.g., autonomous vs. commanded), and length of the mission.

When developing an SRM for a mission, the first step is to identify high-level use cases from the stakeholder's perspective, such as from mission statements. The high-level use case describes the workflow of a business (or operation) goal. Many of the high-level use cases materialize repeatedly in different products. For example, the high-level use cases such as Transport Spacecraft to Destination, Collect Science Data, and Maintain Spacecraft Safety are common goals that appear in almost every NASA science mission. Each of these goal-oriented, high-level use cases are described in detail by a corresponding set of SRM artifacts (e.g., activity diagrams, class diagrams, and statechart assertions).

When a certain use case appears repeatedly in many different missions, then ideally, we would like to create the corresponding set of SRM artifacts for the first mission and reuse it for the other missions. The artifacts would be developed precisely and correctly by an expert SRM modeler and reused by others. In reality, however, without a reuse library and a framework for systematic reuse, the modelers would have to repeat the development process of the artifacts from scratch for every mission with similar use cases. This will necessarily increase the development time and cost and the likelihood of introducing errors in the model.

Our objective, therefore, is to organize common use cases in such a way that will allow the modelers to reuse the corresponding sets of SRM artifacts when they create a new SRM.

To facilitate an effective reuse of SRM artifacts, we propose a classification scheme to categorize use cases into one of the three possible groups: science, spacecraft, and instrument. The science category includes the use cases that pertain to science operations such as transmitting data to earth. The spacecraft category includes the use cases that pertain to spacecraft operations such as putting the spacecraft into an orbit. The instrument category includes use cases that pertain to the instrument operations such as deploying sun shields.

Figure 1 illustrates a single use case diagram for a generic science mission to collect science data that includes use cases from all three categories. We normally use different colors to distinguish categories, but for the diagrams in black-and-white, we use symbols S, V, and M to distinguish the three categories Science, Spacecraft, and Instrument, respectively. The Science category includes operations pertaining to collecting and processing of scientific data. The Spacecraft category includes operations pertaining to maneuvering the spacecraft and maintaining its safety and health. The Instrument category includes operations pertaining to managing instruments for supporting both science and spacecraft operations. A use of an instrument can be strictly for science such as a device for measuring precipitation or strictly for controlling and maintaining a spacecraft such as a solar panel. Some instruments such as antennas can be used in multiple categories.

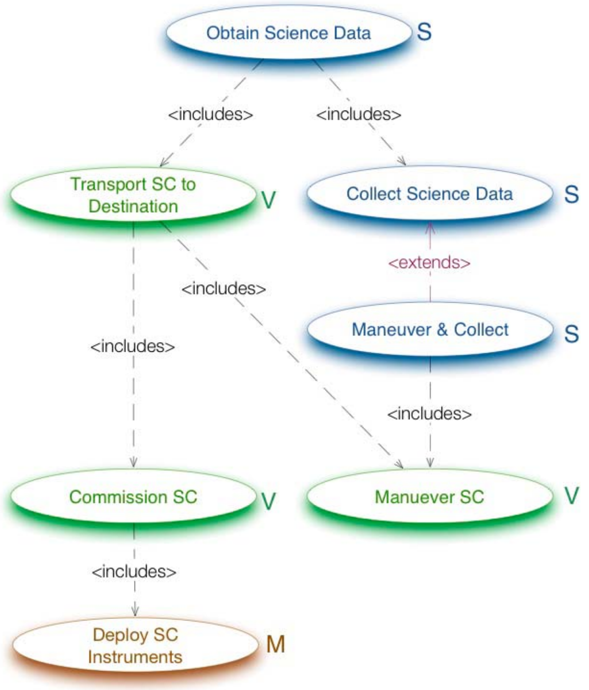


Figure 1: A use case diagram with use cases from all three categories.

In the next section we illustrate how the proposed categorization scheme positively affects the degree of reusability of SRM artifacts.

4. Framework for Goal-based Reuse

In [4], we proposed a framework for the SRM reuse library with a focus on reusing individual artifacts. We describe here how the goal-based reuse increases the unit of reusability from individual artifacts to assets—collections of related artifacts. For the reuse library to be truly useful, it must support both instantiation and adoption reuse. We describe how the types of reuse are achieved in our proposed reuse library by illustrating the reuse library's browsing and searching capabilities. Although browsing and searching can be applied in both types of reuse, we envision browsing as the primary interaction style for instantiation reuse and searching as the primary interaction style for adoption reuse.

Browsing

Browsing can start at any level, but it is typical to start from the topmost level in one of the three categories. Suppose the modeler is interested in a certain type of science operation. The modeler begins browsing by first listing the available topmost use cases in the science operation category, such as *Store Science Data* or *Transfer Science Data*. At the topmost level in each category, there are at most approximately a dozen or so use cases, which is a manageable size for browsing.

Once the modeler locates the desired use case or the one that looks similar to the one he or she is looking for, the modeler can expand the chosen use case by including the related (sub) use cases in the diagram. For example, suppose the modeler determines that it is necessary to maneuver the spacecraft in order to accomplish the goal of obtaining science data, as shown in Figure 1. The modeler can then select the *Maneuver SC* use case and expand it to browse the (sub) use cases as illustrated in Figure 2.

The *Maneuver SC* use case has two sub use cases: *Determine SC Position* and *Maintain SC Stability*. The *Maintain SC Stability* use case in turn has three sub use cases: *Control SC Nutation*, *Control SC Spin Rate*, and *Control SC Precision*. In addition, the *Maneuver SC* use case can be extended to two optional use cases: *Adjust SC Attitude* and *Adjust SC Orbit*, which the modeler may choose to include depending on the spacecraft maneuvering requirements of the science mission.

When expanding the selected use case, the modeler can restrict the expansion to include only those (sub) use cases of the same category. For example, when expanding the *Obtain Science Data* use case of the Science category, the modeler may want to limit the expansion to include only the use cases related to the same Science category. This restriction can be toggled on or off by the modeler while exploring the use cases. The modeler can continue expanding, adding more and more use cases to the level of detail he or she needs to view.

Modelers can follow the hyperlinks to inspect the use case scenarios and other related artifacts. Figure 3 is the domain model that captures essential concepts for the SRM. The links in the diagram depict relationships—connections which the modeler can traverse to view related concepts. The rectangle labeled "Business Goal" contains the goal-based use cases, such as those shown in Figures 1 and 2.

When the modeler finds the use case that most closely matches what he or she wants, the modeler can pull out the associated artifacts of the chosen use case. The associated artifacts include activity diagrams, state diagrams, statechart assertions, etc. Figure 4 shows a sample activity diagram associated with the *Collect Science Data* use case.

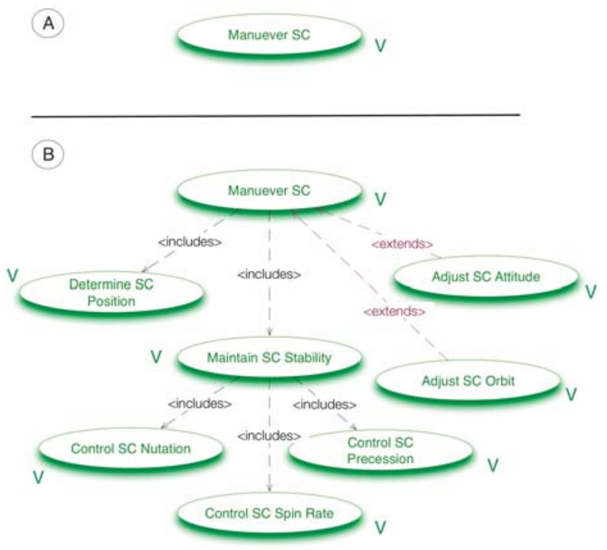


Figure 2: A) A diagram showing the desired (single) use case. B) The expanded version of the same diagram that displays the included (sub) use cases in the same category.

This is an illustration of instantiation reuse, in which the use case is a generic version that captures the common aspects across the different mission types. As such, the associated artifacts are not fully specified. The generic versions include a number of placeholders the modeler has to fill in to construct concrete artifacts for the given project.

Searching

Searching interaction is a desired approach for adoption reuse. The modeler enters values for different

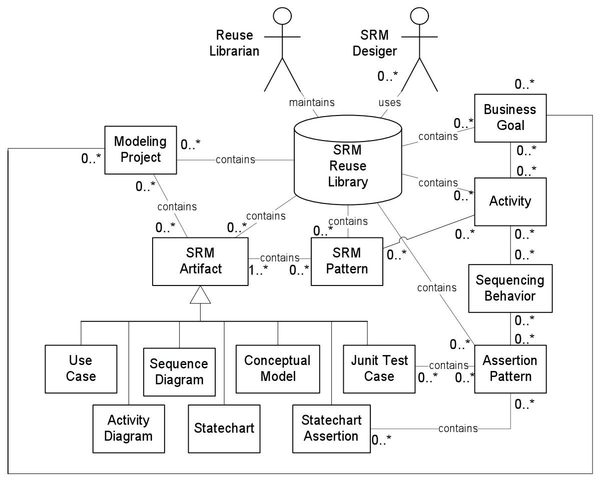
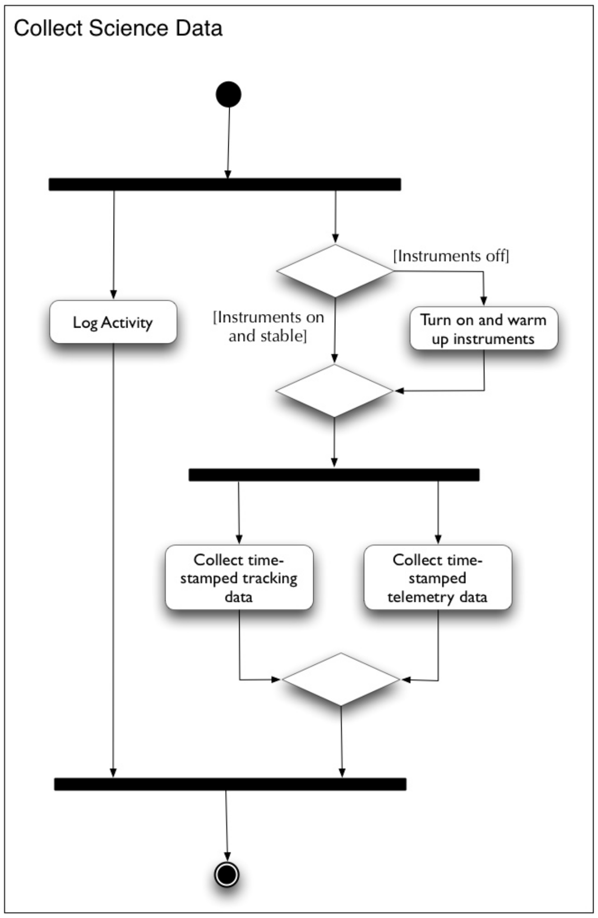


Figure 3: The domain model showing the essential concepts in the SRM reuse library.



Collect Science Data
[Instruments off]
Log Activity
[Instruments on and stable]
Turn on and warm up instruments
Collect time-stamped tracking data
Collect time-stamped telemetry data

Figure 4: A simplified activity diagram for the Collect Science Data use case.

discriminators such as manned or unmanned, orbit type, mission length, and so forth. The system will search the reuse library and retrieve the concrete use cases from the previous missions. The modeler scans through the returned list for partial matches. If the match is close enough, the modeler can retrieve the associated artifacts and adjust them as necessary to fit his or her needs for the mission at hand. If none of the retrieved use cases provide a close enough match then the modeler can browse the generic reuse library for an instantiation reuse or simply search for individual artifacts, as we described in our previous articles [3, 4].

Database Support

To realize the proposed framework we present here, we must implement an effective and efficient database system to manage the artifacts shown in Figure 3. The database system we build should integrate smoothly with the existing tools the modelers use in their modeling work.

It is a well known fact that a relational database management system (RDBMS) is not suitable for an engineering-oriented application domain in which the relationships among different entities are complex. This is because implementing such a database with complex relationships will invariably require a large number of join operations when retrieving data. The number of joins increases in parallel as the length of a navigation path traversing multiple relationships increases. For example, consider retrieving associated statechart assertions for a given business goal in Figure 3. It will require a navigation path of length 4, starting from the Business Goal entity and traversing Activity, Sequencing Behavior, and Assertion Pattern before finally reaching the desired Statechart Assertion entity. Each of these entity types includes a large number of instances, and performing multiple joins on such large sets of data quickly reaches an unacceptable level of performance.

To mitigate a possible performance degradation of the RDBMS for engineering-oriented databases, we anticipate using an object-oriented DBMS for our proposed SRM reuse library. We have not excluded an object-relational DBMS completely, but at this point, we are not considering an object-relational DBMS mainly for two reasons. First, we are not required to connect to any existing relational databases. or create a part of our database in the (pure) relational format. And, second, we believe the additional layer of abstraction for mapping objects into relational tuples and vice versa most likely will become a performance bottleneck for our application. We are in process of investigating candidate object-oriented DBMSs for their suitability in implementing the proposed SRM reuse library.

There is a common set of development tools that modelers routinely use. For our proposed reuse library to be accepted by modelers, it cannot be a standalone system, but instead integrated with that set of commonly used tools. One of those commonly used tools is the Eclipse integrated development environment (IDE) (see http://www.eclipse.org/). We envision building a front-end client module that connects to the SRM reuse library as an Eclipse plugin.

5. Conclusion

We presented our notion of reuse in the context of the System Reference Model (SRM). In this paper, we expanded on our previous research by increasing the unit of reusability from the individual artifacts to collections of related artifacts called assets. For asset-based reusability, we believe the goal-based approach to reuse is most promising. We use a UML use case to describe a goal, and for each use case, there are associated artifacts. The modeler reuses a collection of related artifacts by locating the desired use case. Our proposed reuse library supports both instantiation and adoption reuse of assets. The next step in our research is the detailed design of the proposed reuse library.

6. Acknowledgment

We would like to thank the members of NASA IV&V Facility's SRM Reuse Working Group for the valuable discussions that helped us crystalize the ideas presented in this paper.

7. References

[1] D. Caffall and J. Michael, "Space Applications of System of Systems," in M. Jamshidi, ed., *System of Systems: Principles & Applications*, Boca Raton, Fla.: CRC Press, 2008, pp. 381-397.

[2] D. Drusinsky, J.B. Michael and M. Shing, "A Framework for Computer-Aided Validation," *Innovations in Systems and Software Engineering*, 4(2), June 2008, pp. 161-168.

[3] D. Drusinsky, J.B. Michael, T.W. Otani, and M. Shing, "Validating UML Statechart-Based Assertions Libraries for Improved Reliability and Assurance," *Proc. Second IEEE International Conference on Secure System Integration and Reliability Improvement*, Yokohama, Japan, 14-17 July 2008, pp. 47-51.

[4] T. Otani, B. Michael and M. Shing, "Software Reuse in the IV&V of System of Systems," *Proc. 2009 IEEE International Conference on System of Systems Engineering*, Albuquerque, NM, 1-3 June 2009.

[5] D. Drusinsky, *Modeling and Verification Using UML Statecharts - A Working Guide to Reactive System Design, Runtime Monitoring and Execution-based Model Checking*, Elsevier, 2006.