Modeling Executable Architectural Design Patterns for Software Product Lines

Julie Street Fant[1,2], Hassan Gomaa[1], and Robert G. Pettit[2]

[1]George Mason University
Fairfax, Virginia USA
{jfant, hgomaa}@gmu.edu
[2]The Aerospace Corporation
Chantilly, Virginia USA
{julie.s.fant, robert.g.pettit}@aero.org

Abstract. This paper addresses variability in software product line architectures by addressing variability at a higher level of granularity through architectural design patterns. This approach models three levels of executable architectural design patterns to progressively address variability within the SPL and the member applications. The approach is intended for distributed real-time embedded software domains and has been applied to a space flight SPL.

Keywords: Software Product Lines, software modeling, software architectural design patterns, distributed real-time embedded software, space flight software.

1 Introduction

Many industrial software product lines (SPLs) have significant architectural variability. The engineering of these SPLs emphasize architectural variability at the subsystem, component, and connector levels. For architectures with significant variability, this can become cumbersome due to the multitude of variable components, connectors, and interactions that must be individually modeled. This paper addresses the problem of variability in SPL software architectures by addressing variability at a higher level of granularity through architectural design patterns. Several different combinations of specific components, connectors, and interactions are abstracted into one design pattern. Thus less modeling is required at the SPL level since features have dependencies on design patterns rather than multiple individual components and connectors. The tradeoff is that the application engineering process requires additional modeling since the application specific components, connectors, and interactions must be derived from the design patterns. The key to this approach lies in modeling three levels of architectural design patterns to progressively address variability within the SPL and the member applications, thereby providing the application developer the capability of customizing the patterns to the needs of the application. In addition, developing executable versions of these patterns promote the

validation of the patterns and the architectures they are incorporated into prior to implementation. The approach is intended for distributed real-time embedded (DRE) domains. The three levels of patterns are DRE level patterns, SPL level patterns, and application level patterns. This paper describes how the approach has been applied to and validated on an unmanned space flight software (FSW) SPL.

This paper is organized as follows. First it provides a summary of related work in section 2 and the background on the FSW case study in section 3. Next, the three different levels of architectural design pattern modeling for SPL are described and illustrated using the FSW SPL case study in sections 4, 5, and 6. Finally, validation of the approach is described in section 7 and a discussion on future work in section 8.

2 Related Work

There are many notable SPL approaches including [1], [2], [3]. Other related work includes approaches to build real-time software and embedded architectures from design patterns, such as [3], [4], [5] and software architectural design patterns, such as [6], [7]. These approaches do not explicitly capture variability in the pattern selection for a SPL and SPL member, and do not address variability within design patterns.

This paper builds on the authors' previous work in [3], [8], [9], [10], [11]. In [8] executable design patterns are used to build common features in FSW architectures. This paper extends this work by defining a SPL approach for distributed real-time embedded domains and how to address variability within design patterns. In [9] the feature to design pattern mapping was briefly introduced at a very high level. This paper extends [9] to capture variability in the design pattern selection for SPL members and to provide a more detailed view of feature to design pattern mapping. In [10] the overall approach and its successful application to the FSW domain are described. This paper extends [10] by providing the details on how to address variability within design patterns and design pattern selection. In [11] we compare our pattern based SPL approach with traditional SPL approaches and how it can be applied in the FSW domain. This paper builds on our previous work by emphasizing the architectural views and information captured at each of the three architectural levels.

3 Space Flight Software SPL Case Study

The Space Flight Software (FSW) SPL is used to illustrate the pattern based SPL approach. The FSW SPL involves controlling unmanned spacecraft to achieve its mission. All FSW systems perform similar functions such as command and data handling and attitude control. However, the capabilities of FSW SPL members can vary significantly. For instance, a spacecraft may rely extensively on the ground station to control the spacecraft and require a small amount of hardware to perform its mission. Another spacecraft may utilize onboard autonomy to control the spacecraft and may have extensive hardware to perform its mission. The functionality of other spacecraft can vary between these extremes. Thus the FSW SPL is an ideal domain to apply this pattern-based design approach because of its architectural variability.

4 Distributed RT Embedded Architectural Design Patterns

The highest level of design patterns is the DRE level architectural and executable design patterns. The DRE executable design patterns serve as the foundation for the other two levels of executable design patterns, and are reused across DRE domains. DRE patterns are specified using multiple architectural views. Additionally, an executable version of the design pattern is developed using communicating state machines. This enables designers to validate the design patterns and to validate the executable architectures produced using these design patterns. All modeling is in UML and the executable version is built using IBM Rational Rhapsody.

The first view is a collaboration diagram that captures the components that participate in the design pattern and their variability. At the DRE level, the design patterns are composed of domain components that will be later customized to first the SPL and then the SPL member. Variability is captured using SPL stereotypes from the PLUS method [3]. A collaboration diagram for the Centralized Control design pattern [12] is given in Fig. 1. This depicts the kernel Centralized Controller and optional Input, Output, and IO components, as well as the multiplicity of the associations.

The second architectural view is interaction diagrams, which capture how the objects within the design pattern interact. Because of the wide range of variability in which DRE design pattern can be applied, the interaction diagrams only capture a representative set of object interactions. For instance, the interaction diagram will show the Centralized Controller receiving an input from an Input_Component and in response it will invoke an action on an Output_Component.

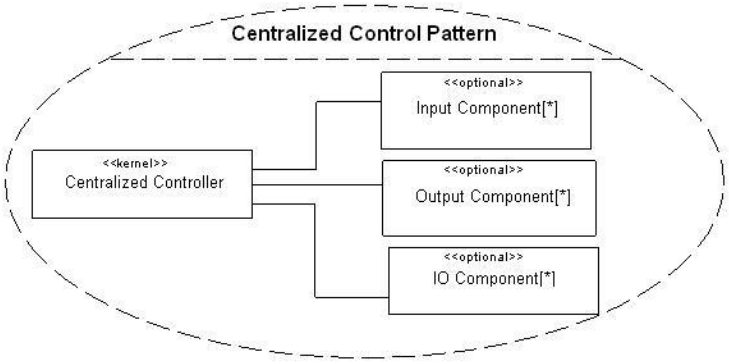

Fig. 1. Collaboration diagram for DRE level Centralized Control executable design pattern

The third architectural view is a component diagram that models the interconnection between components in the DRE design pattern. The component diagram shows the components with required ports, provided ports, and connectors that interconnect the components. Component variability is modeled with SPL stereotypes.

An executable version of the design pattern is also created using communicating state machines. The purpose of the executable version of the design pattern is to specify the internal behavior of a representative set of the pattern's objects and to facilitate validation of the pattern. Each executable design pattern is individually

simulated and validated using Harel's approach of executable object modeling with statecharts [13]. We created a total 21 DRE design patterns [8] using this approach. The approach enables architectures produced by interconnecting these design patterns to be fully executable and validated, as described in Section 7.

A state machine is created for each active component in the design pattern. The state machine captures the dynamic object's internal behavior [13]. To make the state machines executable, the specific states, transitions, actions, and activities that an object performs must be defined. An example of an executable state machine is given for the Centralized_Controller from the Centralized Control design pattern (Fig. 2.). In the Controlling state, the input event is received from the input component and in response a command is sent to the appropriate output or I/O component.

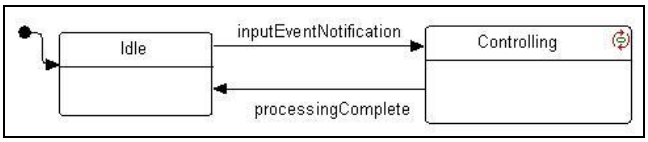

Fig. 2. State machine for Centralized_Controller component

5 SPL Architectural Executable Design Patterns

The next level of design patterns is at the SPL level; these patterns are created during the domain engineering phase. The purpose of creating SPL level design patterns is to add SPL domain knowledge and variability to the DRE design patterns and to create SPL architectures from the SPL design patterns. This section describes creating SPL design patterns for the FSW SPL.

5.1 SPL Pattern-based Feature Modeling

Features represent common and variable characteristics or requirements of the SPL. Features are analyzed and categorized as common, optional, or alternative. Related features can be grouped into feature groups, which constrain how features are used by a SPL member [3]. The feature model for a pattern-based SPL architecture needs to incorporate pattern-related features, so that specific patterns can be selected during application engineering. Features that are mapped to variable design patterns are called pattern specific features. *Pattern specific features* are coarse grained features that relate to a design pattern and differentiate among other related features. *Pattern variability features* are fine grained features, which define the variability within a pattern specific feature. Alternative pattern specific features can be grouped into an exactly-one-of feature group, one of which must be selected for a given application.

When feature modeling was applied to the FSW SPL Command and Data Handling (C&DH) subsystem, 52 features were identified. A subset of the FSW C&DH feature model is shown in Fig. 3. This feature model contains an «exactly-one-of feature group» called Command Execution, which addresses how spaceflight commands are processed. This feature group has three «alternative» features. The Low Volume

Command Execution feature is used when a small amount of commands needs processing, the High Volume Command Execution feature is used when a large amount of commands needs processing, and Time Triggered Command Execution is used when commands must be executed with strict temporal predictability.

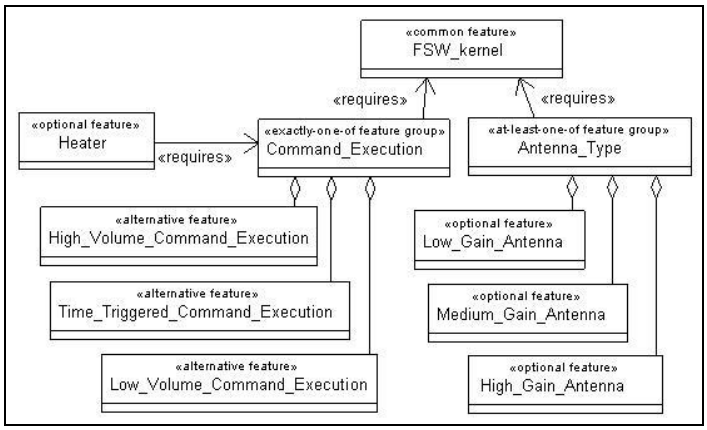

Fig. 3. Subset of FSW C&DH feature model

There is also a significant amount of variability in the amount and type of hardware that must be commanded, which are captured in variation points. The optional Heater feature influences whether the FSW is required to execute commands to a heater. Finally, the last feature group in Fig. 3 is the Antenna Type «at-least-one-of feature group». This group has three optional features and minimally one must be selected since all FSW have at least one antenna for communicating with the ground station.

The next step is to create a feature to design pattern mapping. The purpose of the feature to design pattern mapping is to determine which variable design patterns could be mapped to SPL features. This step uses behavioral interaction modeling, as described later. When this step was applied to the FSW SPL, a SPL interaction model was created for each of the pattern specific features, 24 in total.

5.2 SPL Architectural Executable Design Patterns

Next, the general purpose components in the DRE architectural design pattern are updated to be SPL specific. This includes identifying SPL components, their multiplicities, and variability. This process is illustrated using the DRE Centralized Control design pattern (Fig. 1.) in the FSW SPL (Fig. 4.). This design pattern captures the I/O devices the FSW interacts with based on the ground commands it receives. To realize this mapping, the Centralized Controller and the I/O components in Fig. 1. are updated to give the CDH Centralized Controller and FSW I/O components in Fig. 4. SPL components are categorized as kernel, optional, or variant.

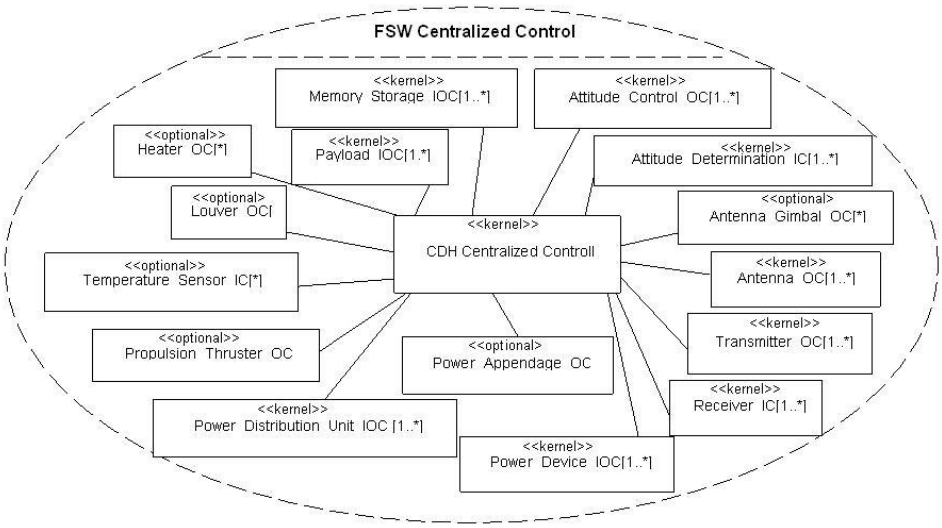

Fig. 4. FSW SPL Centralized Control collaboration diagram

For example, the SPL requires ground commands that adjust the spacecraft's attitude by invoking actions on attitude control devices. This is achieved by updating an Output_Component in the DRE level design pattern to become an Attitude_Control_Device_OC kernel component. Since there can be different versions of attitude control in different missions, different attitude control devices are modeled as variants. Additionally, since a heating device is optional, another Output_Component is updated to an optional Heater_OC. The Centralized Control design pattern is only used when there are a small amount of commands to execute and hardware to control. Therefore, when this pattern is customized to an application, only an application specific subset of the optional devices is selected.

Next, the interaction diagrams must be updated to reflect the needs of the SPL. If the precise sequence of object interactions is known, then it should be modeled. However, in design patterns where there is high amount of variability in the object interactions, then only a representative set of object interactions is modeled. In that case, detailed interaction modeling is deferred to application engineering. For the FSW SPL's C&DH subsystem, 24 interaction diagrams were created, one for each of the pattern specific features described in section 5.1.

Next, the executable version of the design pattern must also be updated for the SPL. For the FSW SPL's C&DH subsystem, a state machine was created for each active component in the FSW SPL's 24 patterns. This involves potentially adding SPL specific states, actions, and activities to the existing DRE level state machines. For example, if the SPL pattern specific behavior needs to be added, then this can be modeled in substates. If a component needs to send a message to another component, then this is modeled as an action. Finally, other application specific logic, such as a processing algorithm, can be model as activities. This step is illustrated using the state machine for the CDH_Centralized_Controller from the Centralized Control De-

sign pattern (Fig. 5), which is derived from the DRE level Centralized_Controller component (Fig. 2.). A requirement of the SPL feature is for this component to manage and account for the spacecraft mode. Therefore the common modes, including launch, normal, and safe modes are added to the highest level in the state machine, as seen Fig. 5. Within each of the modes are Idle and Controlling states from the original Centralized Control state machine. According to the feature, all ground commands or responses to onboard events must be validated and log any commands executed or rejected. Since this is a refinement, this behavior is modeled as substates within the Controlling states, as shown in Fig. 5.

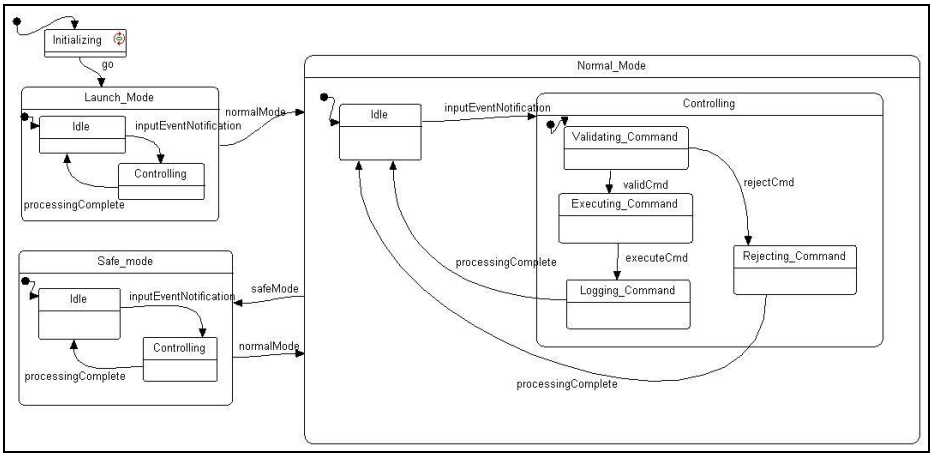

Fig. 5. State machine for FSW SPL CDH_Centralized_Controller

6 Application Engineering

After the FSW SPL is complete, applications can be derived from this FSW SPL architecture. The process is applied to the Student Nitric Oxide Explorer (SNOE) and Solar TErrestrial RElations Observatory (STEREO) application case studies, which are real-world space programs [14], [15]. SNOE mission involves using a small spin stabilized spacecraft in a low earth orbit to measure thermospheric nitric oxide and its variability. SNOE is a low earth orbit and relies heavily on the ground station to control the spacecraft's small amount of hardware. STEREO mission involves using two nearly identical three-axis stabilized spacecraft orbiting around the sun to study the studying the nature of coronal mass ejections. Since STEREO is not in constant communication with the ground station, it relies on a significant amount of autonomy and stored ground commands to control the spacecraft. These case studies were selected because they cover a wide variety of spacecraft in the FSW domain.

This paper describes application engineering for SNOE, in which application features are selected based on SNOE's requirements. For example, as SNOE is only required to process a low volume of ground commands, the Low Volume Command Execution feature is selected from the Command Execution feature group in Fig 3.

The final level of design patterns are at the application level, which are created during application engineering. The application's executable design patterns are derived from the SPL executable design patterns. This involves customizing the SPL design pattern specification and executable version based on the feature to design pattern mapping and feature selection for this application. SNOE uses the Centralized Control design since it is mapped to its Low Volume Command Execution feature.

First, the collaboration diagram is customized to reflect the application specific components. This involves removing optional components that are not selected, updating the component multiplicities, and selecting the appropriate variants based on the application's features. In some cases variants are unique to the application, in which case they must be defined at the application level. This process is illustrated using SNOE's Centralized Control executable design pattern collaboration diagram shown in Fig. 6, which is derived from the FSW SPL collaboration diagram (Fig. 4). Based on SNOE's feature selection, it is determined that none of the optional components are utilized and are removed. Additionally, the SNOE specific variants are selected based on SNOE's feature selection. For example, instead of using Antenna_OC (Fig. 4), the Low_Gain_Antenna_OC variant, is used (Fig. 6). SNOE contains application unique payloads variants, such as Auroral_Photometer_IOC, and microGPS_IOC, which are are added to the collaboration diagram.

Next, the interaction diagrams must be customized for the application based on the feature selections for the application. Now, the specific set of object interactions is modeled rather than just a representative set. For instance, consider an application specific sequence for SNOE's Centralized Control interaction diagram is based on the FSW SPL interaction diagram. However the application specific variants and specific interactions are used, rather than a just a representative set of interactions.

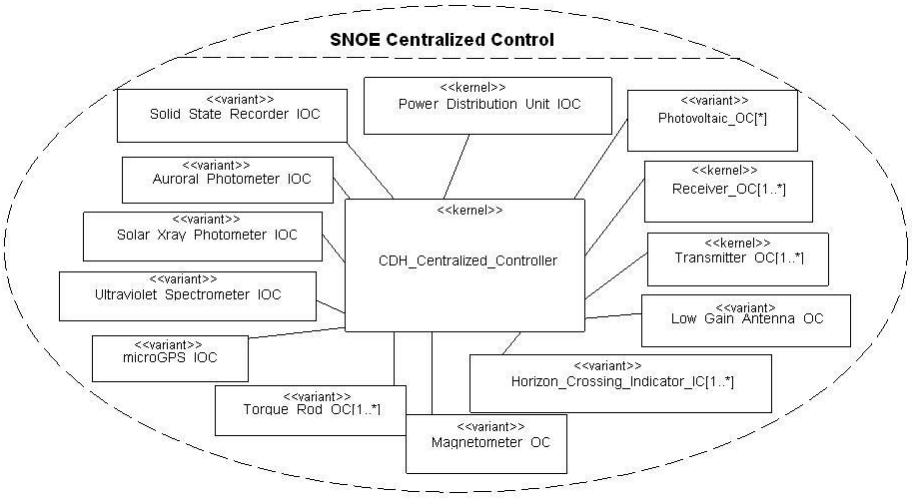

Fig. 6. SNOE specific Centralized collaboration diagram

Next, the executable versions of the design patterns were updated for the application. This involves potentially adding application specific states, actions, and activities to the SPL level state machines based on the application's features. For example, if the application features refines some behavior, then this can be modeled as substates. If the component must send a message to an application specific variant or if application specific logic is required then this is modeled as an action or activity within a state or transition. This process is illustrated using the state machine for the SNOE's CDH_Centralized_Controller, which is a customized version of the FSW SPL's CDH_Centralized_Controller (Fig. 5). SNOE's features require application specific logic to validate commands, to determine the appropriate response to commands, and to interface with SNOE unique payload variants. This information is modeled as actions and activities within the states, which are updated in the appropriate states.

7 Validation

The approach to validate the DRE patterns, the FSW product line, and the SNOE and STEREO application case studies involved several validation steps throughout the development. First, the individual DRE design patterns were validated by ensuring functional correctness of the individual executable design patterns. This was accomplished by creating test cases to cover all states, transitions, and actions for the state machines of all the components in the DRE executable design pattern. Input data to test cases included source states and event sequences that trigger a test case and output data including expected destination states and actions, as described in [8].

Second, the FSW SPL individual design patterns were also validated for functional correctness. Again, test cases were created that covered all states, transitions, and actions for the state machines of all the components. Then the expected results of the test cases were compared with the actual behavior of the state machines.

Thirdly, the SNOE and STEREO design patterns were individually validated. Again, test cases were created to cover all states, actions, and transitions for the design patterns. However, test cases are different from the FSW SPL test cases because they must test all of the application customizations, including data, logic, and additional states. Then the test cases were compared with the actual behavior of the state machines. Finally, the entire SNOE and STERO architectures, including the design pattern interconnections, were validated. To achieve this, a feature based validation approach based on CADeT [16] was applied. This approach helps to reduce the overall validation effort by created reusable SPL test assets that can be customized for SPL applications. This testing is different from validation of the individual design patterns because it tests how the design patterns are integrated together. A total of 22 feature-based test specifications were created and passed for SNOE and 32 feature-based test specifications were created and passed for STEREO.

8 Conclusions and Future Work

This paper has described an approach for addressing SPL architectural variability at a larger degree of granularity using software architectural design patterns. The key to this approach lies in modeling three levels of these patterns to progressively address

variability within the patterns themselves and variability in the patterns selected for a member application. In addition, developing executable versions of these patterns allows the validation of the patterns and the architectures they are incorporated into. The approach described in this paper has several benefits. First, the approach provides DRE patterns that can be applied in DRE product lines and systems. Second, executable versions of the patterns are developed to allow the patterns to be validated. Third, providing three levels of patterns provides a systematic approach for incorporating executable architectural design patterns into the SPL architecture and member applications. Furthermore, executable architectures produced from these patterns can be validated during the design phase for functional correctness. For future work, this research can be applied to other DRE SPLs and to other application domains.

References

[1] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, 2002.

[2] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering Foundations, Principles, and Techniques*. Springer, 2005.

[3] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison-Wesley , 2005.

[4] B. Selic, "Architectural Patterns for Real-Time Systems: Using UML as an Architectural Description Language," in *UML for Real*, Springer, 2004, pp. 171–188.

[5] D. Bellebia and J.-M. Douin, "Applying patterns to build a lightweight middleware for embedded systems," Proc. Pattern languages of programs Conf., 2006.

[6] E. Gamma, R. Helm, R. Johnson, and V. John, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1995.

[7] R. Pettit IV and H. Gomaa, "Modeling Behavioral Design Patterns of Concurrent Objects," in *International Conference on Software Engineering (ICSE)*, China, 2006.

[8] J. Fant, H. Gomaa, and R. Pettit, "Architectural Design Patterns for Flight Software," *IEEE Wkshp on Model-based Eng. for RT Embedded Systems*, CA, 2011.

[9] J. Fant, "Building Domain Specific Software Architectures from Software Architectural Design Patterns," ICSE ACM Student Research Competition, Hawaii, 2011.

[10] J. S. Fant, H. Gomaa, and R. Pettit, "Software Product Line Engineering of Space Flight Software," in *3rd Int. Wkshp on Product LinE Approaches in Softwr Eng* 2012.

[11] J. S. Fant, H. Gomaa, and R. Pettit, "A Pattern-based Modeling Approach for Software Product Line Engineering," in *46th Hawaii Intl. Conf. on System Sciences* 2013.

[12] H. Gomaa, *Software Modeling and Design: UML, Use Cases, Architecture, and Patterns*. Cambridge University Press, 2011.

[13] D. Harel, "Executable object modeling with statecharts," presented at the 18th International Conference on Software Engineering, Berlin,1996.

[14] Laboratory For Atmospheric and Space Physics, University of Colorado,Boulder, "Student Nitric Oxide Explorer Homepage," *http://lasp.colorado.edu/snoe*.

[15] Johns Hopkins University Applied Physics Laboratory, "STEREO Web Site," 26-Apr-2010. [Online]. Available: http://stereo.jhuapl.edu/index.php.

[16] E. M. Olimpiew and H. Gomaa, "Reusable Model-Based Testing," Proc 11th International Conference on Software Reuse, Falls Church, VA, Sept. 2009.