Building Domain Specific Software Architectures from Software Architectural Design Patterns

Julie Street Fant
George Mason University
4400 University Drive
Fairfax, Virginia 22030
jstreet1@gmu.edu

ABSTRACT
Software design patterns are best practice solutions to common software problems. However, applying design patterns in practice can be difficult since design pattern descriptions are general and can be applied at multiple levels of abstraction. In order to address the aforementioned issue, this research focuses on creating a systematic approach to designing domain specific distributed, real-time and embedded (DRE) software from software architectural design patterns. To address variability across a DRE domain, software product line concepts are used to categorize and organize the features and design patterns. The software architectures produced are also validated through design time simulation. This research is applied and validated using the space flight software (FSW) domain.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures – *domain specific architectures, languages, patterns.*

General Terms
Design

Keywords
UML, Software Architectures, Design Patterns, Distributed Real-Time and Embedded Software, Space Flight Software.

1. INTRODUCTION
Software design patterns are best practice solutions to common software problems. Design patterns are typically presented at the domain independent level. Capturing design patterns in this fashion makes them applicable across multiple domains and also at multiple levels of abstraction. However, the generic nature of the design patterns make them difficult to adopt in practice because it is not always clear how they should be applied in a given domain. This research addresses this problem by focusing on a systematic approach to designing domain specific DRE software from software architectural design patterns. As part of this approach, the software architectures produced are also validated through design time simulation.

ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA

This research is applied and validated using the space flight software domain. FSW is an ideal domain to apply this research. First, there is an industry trend on the growing number of software related spacecraft anomalies. In fact, "in the period between 1998 and 2000 nearly half of all observed spacecraft anomalies were related to software" [1]. This research can help to alleviate the number of software related anomalies by providing design time validation through simulation. Therefore, design flaws that lead to software anomalies can be identified and remedied early in the software lifecycle. Additionally, this research aligns with an industrial recommendation from the National Aeronautics and Space Administration (NASA) to help improve FSW acquisitions with early analysis and architecting of FSW [2].

2. BACKGROUND AND RELATED WORK
The idea of building software architectures from software design patterns is not new. There are many notable approaches for building software architectures from design patterns [3-7] and more specifically real-time systems [8-11]. These approaches only provide abstract descriptions of design patterns and high level guidance on how patterns can be used to form software architectures. Other researchers are developing approaches to help aid practitioners in design pattern selection [12-15]. Finally, other research focuses on increasing design pattern application using new teaching methods [16-20]. None of the mentioned related works provide detailed guidance for applying design patterns across a specific domain nor do they provide executable templates to make applying the patterns easier.

3. APPROACH AND UNIQUENESS
This research describes a systematic approach for designing domain specific DRE software from software architectural design patterns. The foundation for this approach involves using a feature model, feature to design pattern mapping, and executable design pattern templates. These are used to structure the design patterns for a specific DRE domain. The systematic approach for developing and applying feature model, feature to design pattern mapping, and the executable design pattern templates for a specific DRE domain is described below in more detail.

The first step in applying this research to a particular DRE domain involves performing domain analysis. Domain analysis is performed to identify the common and variable features. A feature is defined as a requirement or characteristic of some systems in a given DRE domain. A software product line (SPL) feature model [7] is used to identify the variability and interrelationships among the features. Variability is captured using SPL stereotypes to indicate whether a feature is common to all applications or only provided by some applications in a given

domain. For example, <<kernel>> is used to indicate a feature is common to all applications and <<optional>> is used to describe a feature that is provided by some applications.

Second, each of the domain features is mapped to certain design patterns that support that feature. The result is a domain feature to design pattern mapping. Since design patterns are general purpose it is possible that they can be mapped to multiple different features. The feature to design pattern mapping can be illustrated using the FSW domain's format telemetry <<kernel>> feature. This feature involves processing spacecraft data into telemetry packets that are ready for transmission. This feature is mapped to the pipes and filters, master slave, and strategy design patterns since they can be utilized to achieve format telemetry <<kernel>> feature.

The feature to design pattern mapping is where this research is very novel and differentiates itself from transitional SPL approaches. Traditional SPLs sometimes utilize a feature to class mapping that aids the engineer in selecting the specific classes required for a SPL member [7]. This approach works very well when SPL members share architectural similarities. However, traditional SPL approaches do not work well across a domain because they do not provide architectural flexibility. For example, a SPL for a small spacecraft's FSW may map the control feature to a centralized controller class. However, this SPL architectural foundation would not suffice for a large spacecraft's FSW because the centralized controller class would quickly become a bottleneck and render the spacecraft useless. The feature to design pattern mapping does provide this architectural flexibility. For example, the FSW domain's main control feature can be mapped to the centralized control, hierarchical control, and distributed control design patterns. Therefore when a small spacecraft's FSW is designed the centralized control pattern can be selected and the hierarchical control pattern can be selected when a large spacecraft is architected.

The last step in the proposed approach is to build domain specific executable design pattern templates are created for each of the design patterns. The templates capture the design pattern along with any domain specific knowledge. The executable design pattern templates provide a specific structure to the design patterns and are used to make the design patterns easier to apply. The templates are captured in the Unified Modeling Language (UML) and consist of both static and dynamic views. UML is used because it is the de facto modeling language of object oriented systems. The executable nature of the templates lies in the state machines. A state machine is created to capture the internal behavior of each concurrent component in the design pattern. The state machines are executed using executable statecharts semantics [22].

Finally, the approach can then be applied to build a specific application. This systematic process involves selecting the appropriate features from the feature model. Then design patterns to achieve these features are selected from the domain feature to design pattern mapping. Next, the executable templates are instantiated for the particular application and application specific information is added to the templates. Finally, the templates are interconnected with the rest of the architecture. This involves interconnecting the structural patterns through communication patterns. The resulting software architecture can then be validated using executable statecharts.

This research is unique because it goes beyond providing guidance on how design patterns can be used to build domain specific DRE software architectures. It takes the additional step of describing how design patterns can be applied to a particular domain using feature models and feature to design pattern mappings. The feature model leverages SPL concepts to address variability and the feature to design pattern mapping gives this approach the flexibility to address architectural variability across a domain. Additionally, it provides executable templates to make applying the patterns easier. Finally, this research also facilitates design time validation of the software architectures produced because the resulting architectures are also executable. The architecture can then be validated by tracking the flow of events and messages through the architecture to ensure functional correctness.

4. VALIDATION AND RESULTS

This research is validated by applying it to the FSW domain and building two real world FSW case studies. The case studies include the command and data handling (C&DH) subsystems from a small, spin stabilized spacecraft in a low earth orbit and a large, three-axis stabilized spacecraft in heliocentric orbit. These case studies were selected because they illustrate the feasibility of applying this research to wide range of FSW.

First, domain analysis is performed on FSW C&DH subsystems. Analysis reveals that there are eight <<kernel>> features and three <<optional>> features. The interrelationships between these features are captured in a feature model. Then, each of the FSW features is mapped to certain design patterns that support the feature. For example, consider the command execution <<kernel>> feature. This feature involves determining when and which spacecraft commands should be executed. The centralized control, hierarchical control, distributed control, command, and command dispatcher design patterns can be utilized to achieve this feature. Therefore they are mapped the command execution <<kernel>> feature. For the C&DH subsystem a total of 32 unique architectural design patterns are mapped to the 11 main features.

Next, the executable design pattern templates are also created for the FSW domain. For example, common steps involved in the command execution <<kernel>> feature include validating the command, executing the command, logging the command, and rejecting the command. These steps can be included in certain design patterns associated with the command execution <<kernel>> feature. The process for deriving these executable templates for the FSW domain is described in more detail in [21].

Finally, the approach is applied to build the two case studies. The approach successfully produced executable C&DH architectures from software architectural design patterns for both of the case studies. Since the case studies covered a wide range of functionality, they exercised a variety of the <<optional>> features from the feature model and different design pattern combinations from the FSW feature to design pattern mapping. The C&DH patterns and architectures produced are also validated using executable statecharts in IBM Rational Rhapsody.

5. CONTRIBUTIONS

The main contribution of this research is an approach for building domain specific DRE software architectures from software architectural patterns. This approach improves the quality of DRE software architectures by leveraging the benefits of software design patterns. This approach is flexible enough to address a wide variety of architectures using a single domain specific feature to design pattern mapping. This ultimately will save engineers when multiple different applications within a domain are required. Additionally, the executable design pattern templates not only save an engineer time when building software architectures, they also provide the foundation for performing design time validation on the software architecture produced using this approach.

However, this approach does have limitations. First, similar to traditional SPL approaches, it requires additional upfront engineering to build the feature model and the feature to design pattern mapping. Additionally, the process to build the initial executable templates is manual. Therefore this approach is not appropriate for building single applications within a domain. The main benefits are realized when multiple different applications are required.

Second, the feature to design pattern mapping does not provide any rules for pattern combinations that can and cannot be applied. For example, the FSW command execution <<kernel>> feature is mapped to the centralized control, hierarchical control, distributed control, command, and command dispatcher design patterns. However, some pattern combinations like centralized control and distributed cannot be used together. Therefore future research will develop the rules for integrating design patterns.

Finally, while industrial software engineering tools can be used to support the proposed approach, applying the research is still a manual process. Thus future research will also look for ways to automate the process and the application of design patterns.

6. ACKNOWLEDGEMENTS

The author would like to thank Dr. Hassan Gomaa for his advice and guidance of this research. This research is based on work supported by The Aerospace Corporation.

7. REFERENCES

[1] M. Hecht and D. Buettner, "Software Testing in Space Programs," *Crosslink*, vol. 6, no. 3, 2005.

[2] D. Dvorak (editor), *NASA Study on Flight Software Complexity*. NASA Office of Chief Engineer, 2009.

[3] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages*. Hoboken, NJ: John Wiley & Sons, LTD, 2007.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Hoboken, NJ: John Wiley & Sons, LTD, 1996.

[5] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*. Hoboken, NJ: John Wiley & Sons, LTD, 2000.

[6] M. Kircher and P. Jain, *Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management*. Hoboken, NJ: John Wiley & Sons, LTD, 2004.

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

[8] B. Douglass, *Real-Time Design Patterns*. Addison-Wesley, 2003.

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

[10] J. Zalewski, "Real-Time Software Architectures and Design Patterns: Fundamental Concepts and Their Consequences," *Annual Reviews in Control*, vol. 25, 2001.

[11] I. Fliege, A. Geraldy, R. Gotzhein, T. Kuhn, and C. Webel, "Developing safety-critical real-time systems with SDL design patterns and components," *Computer Networks*, vol. 49, 2005.

[12] J. Noble, "Classifying Relationships Between Object-Oriented Design Patterns," in *Australian Software Engineering Conference*, 1998.

[13] B. Schulz, T. Genbler, B. Mohr, and W. Zimmer, "On the Computer Aided Introduction of Design Patterns into Object-Oriented Systems," in *Technology of Object-Oriented Languages and Systems (TOOLS)*, 1998.

[14] L. C. Briand, Y. Labiche, and A. Sauvé, "Guiding the Application of Design Patterns Based on UML Models," in *22nd IEEE International Conference on Software Maintenance (ICSM'06)*, 2006.

[15] K. Meffert, "Supporting Design Patterns with Annotations," in *13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'06)*, 2006.

[16] A. Oluyomi, S. Karunasekera, and L. Sterling, "An Agent Design Pattern Classification Scheme: Capturing the Notions of Agency in Agent Design Patterns," in *11th Asia-Pacific Software Engineering Conference (APSEC'04)*, 2004.

[17] K. S. Y. Al-Tahat, A. N. Al-ahmad, N. B. Kallajo, and W. B. Al-Jayyousi, "A Design Pattern Management Tool for Educational Purposes," in *The 2nd Information and Communication Technologies (ICTTA)*, 2006.

[18] C. Alphonce, M. Caspersen, and A. Decker, "Killer "Killer Examples" for Design Patterns," in *SIGCSE'07*, 2007.

[19] P. V. Gestwicki, "Computer Games as Motivation for Design Patterns," in *SIGCSE'07*, 2007.

[20] D. R. Wright, "The Decision Pattern: Capturing and Communicating Design Intent," in *SIGDOC'07*, 2007.

[21] J. Fant, H. Gomaa, and R. Pettit IV, "Architectural Design Patterns for Flight Software," in *2nd IEEE Workshop on Model-based Engineering for Real-Time Embedded Systems*, 2011.

[22] D. Harel, "Executable object modeling with statecharts," in *18th International Conference on Software Engineering*, 1997