An Outline for a Domain Specific Software Architecture Engineering Process

Will Tracz
Lou Coglianese
IBM Corporation
Federal Sector Division
MD 0210
Owego, NY 13827-1298

Abstract

"In order to reuse software, there needs to be software to reuse."

One of the dilemmas that has prevented software developers from reusing software is the lack of software artifacts to use or the existence of artifacts that are difficult to integrate. Domain Specific Software Architectures (DSSAs) have been proposed in order to address these issues. A DSSA not only provides a framework for reusable software components to fit into, but captures the design rationale and provides for a degree of adaptability. This paper presents an outline for a Domain Specific Software Architecture engineering process.

Keywords: Domain Analysis, Domain Specific Software Architecture, Domain Engineering

1 Introduction

The purpose of this paper is to outline a process[1] currently under development, that can be used to generate a Domain-Specific Software Architecture (DSSA). It is based on the work of Ruben Prieto-Diaz[3] and Sholom Cohen[1]. The fundamental premises of this work are that:

1. an application can be defined by a set of needs that it fulfills,
2. user needs can be mapped into a set of requirements that meet those needs,
3. requirements can be met in a number of ways,
4. implementation constraints limit the number of ways requirements can be met.

[1]Note: a process is a series of steps or stages with entry and exit criteria and tasks to follow at each phase as well as a way on verifying if what you are doing is any good.

The goal of the process is to map user needs into system and software requirements that, based on a set of implementation constraints, define a DSSA. The separation of user needs from system requirements and implementation constraints differentiates this process from previous work. In particular most domain analysis processes do not differentiate between functional requirements and implementation constraints, but rather simply classify them under the heading of "requirements". Similarly, this process differentiates between the System Architecture and the Software Architecture that is part of it.

Another difference between this approach to domain engineering and other domain analysis approaches (e.g., Prieto-Diaz[2]) is that case-based reasoning and reverse engineering are not central mechanisms for identifying reusable resources, but rather existing applications are used as vehicles to validate the architectures that are derived, top-down, from generalized user requirements[2].

At the top-most level there are 5 stages in the process. Each stage is further broken into sub-steps or stages. Furthermore, this process is concurrent, recursive, and iterative, therefore completion may require several passes through each stage with additional levels of detail being addressed, or new insights (or oversights) requiring further definition or analysis. For example, during **Stage 1: Defining/Scoping the Domain**, one is concurrently identifies key aspects of the domain, which is part of **Stage 2.2: Defining a Domain Vocabulary**.

The five stages in the DSSA Definition Process are:

1. **Define/Scope the Domain**
   - Define what can be accomplished — emphasis is on user's needs.
2. **Define/Refine Domain Specific Concepts/Requirements**
   - Similar to Requirements Analysis — emphasis is on problem space.
3. **Define/Refine Domain Specific Implementation Constraints**
   - Similar to Requirements Analysis — emphasis is on solution space.
4. **Develop Domain Architectures/Models**
   - Similar to High-Level Design — emphasis is on defining module/model interfaces and semantics.
5. **Produce Reusable Workproducts**
   - Implementation of reusable artifacts (e.g., code, documentation, etc.).

The remaining material in this paper consist of a breakdown of the stages listed above. A detailed description of each stage is found in[4], which is currently under development. Each stage consists of a series of questions to be answered and a list of outputs to be generated.

[2]The reuse of existing artifacts is not the central goal of the proposed Domain Engineering process, but rather the development of a reusable architecture into which, well-specified components can be integrated.

1.1 Domain Engineering Process Overview

The proposed Domain Engineering process consists of the following steps:

Stage 1 Define/Scope the Domain

- **Stage 1.1** Define goals of domain modeling.
- **Stage 1.2** Define the domain.
  - **Stage 1.2.1** Identify what is **inside** the domain.
  - **Stage 1.2.2** Identify what is **outside** the domain.
  - **Stage 1.2.3** Identify what is on the **borders** of the domain (input/output).
- **Stage 1.3** Define Domain Specific Resources
  - **Stage 1.3.1** Define **who** you have to work with.
  - **Stage 1.3.2** Define **what** you have to work with.
  - **Stage 1.3.3** Define **how** you will verify the models.
- **Stage 1.4** Define the **domain of interest** (subset of work that could be done).

Stage 2 Define/Refine Domain Specific Concepts/Entities/Requirements

- **Stage 2.1** Define a block diagram architecture (E/R Diagram).
  - **Stage 2.1.1** Identify concepts/entities (behavior, temporal, and data) in the domain.
  - **Stage 2.1.2** Identify attributes of concepts/entities.
    - **Stage 2.1.2.1** Identify required/essential/mandatory concepts/entities.
    - **Stage 2.1.2.2** Identify optional concepts/entities.
    - **Stage 2.1.2.3** Identify alternative concepts/entities.
    - **Stage 2.1.2.4** Identify requirements common from application to application
    - **Stage 2.1.2.5** Identify requirements that vary from application to application.
  - **Stage 2.1.3** Identify relationship between concepts/entities.
    - **Stage 2.1.3.1** Identify "is a/a kind of" relationships between concepts.
    - **Stage 2.1.3.2** Identify "consists of" relationships between concepts.
    - **Stage 2.1.3.3** Identify "uses/needs" relationships between concepts.
  - **Stage 2.1.4** Classify and cluster common concepts.
  - **Stage 2.1.5** Record issues, trade-offs, and rationale.
  - **Stage 2.1.6** Create a Domain Description Document.
- **Stage 2.2** Create a Domain Vocabulary Dictionary.
  - **Stage 2.2.1** Create a Domain Thesaurus (list of synonyms).
- **Stage 2.3** Create a High-Level Requirements Specification document.
- **Stage 2.4** Refine concepts/entities already identified to reflect desired level of detail.

Stage 3 Define/Refine Domain Specific Implementation Constraints.

Stage 3.1 Define general implementation constraints on the architecture.

Stage 3.1.1 Define general software constraints (e.g., programming language, run-time system, or Operating System)

Stage 3.1.2 Define general hardware/physical constraints (e.g., platform, sensors, or appearance).

Stage 3.1.3 Define general performance constraints (e.g., response time, accuracy).

Stage 3.1.4 Define general mission constraints (e.g., fault tolerance, security, or safety).

Stage 3.2 Identify relationships between concepts and constraints.

Stage 4 Develop Domain Architectures/Models.

Stage 4.1 Define a Domain Specific Software Architecture or Architectures.

Stage 4.1.1 Define a decision taxonomy for requirements and constraints for each architecture.

Stage 4.1.2 Record design issues, trade-offs, and decision rationale.

Stage 4.2 Specify interfaces for each module (operations and operands).

Stage 4.2.1 Specify semantics of each module (behavior).

Stage 4.2.2 Specify constraints on each module (e.g., entry/exit criteria).

Stage 4.2.2.1 Specify performance/timing constraints.

Stage 4.2.2.2 Specify dependency (layering) constraints.

Stage 4.2.2.3 Specify sequentiality/order (operational) constraints.

Stage 4.2.2.4 Specify mission constraints.

Stage 4.2.3 Specify performance characteristics of each model.

Stage 4.2.4 Identify configuration (generic) parameters for each model.

Stage 4.2.5 Record issues, trade-offs and design rationale.

Stage 4.3 Link models to concepts and requirements.

Stage 4.4 Refine a Domain Specific Software Architecture or Architectures to reflect desired level of detail.

Stage 5 Produce Reusable Workproducts.

Stage 5.1 Develop the reusable artifacts (e.g., code, documentation, test cases, etc.)

Stage 5.1.1 Determine parameterization/configurability level desired.

Stage 5.1.2 Implement each module.

Stage 5.1.3 Test each module.

Stage 5.1.4 Document each module.

Stage 5.1.5 Record issues, trade-offs and design rationale.

Stage 5.2 Link artifacts to models, concepts, and requirements.

Stage 5.3 Link documentation to models, concepts, and requirements.

References

[1] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, November 1990.

[2] R. Prieto-Diaz. Domain Analysis for Reusability. In *Proceedings of COMPSAC 87*, 1987.

[3] R. Prieto-Diaz. Reuse Library Process Model. Technical Report AD-B157091, IBM CDRL 03041-002, STARS, July 1991.

[4] W. Tracz and L. Coglianese. Domain Engineering Process Guidelines. Technical Report TBD, IBM Federal Sector Division, In Progress 1991.

About the Authors

Will Tracz

Will Tracz is a senior programmer at the Owego Laboratory of the IBM Federal Sector Division specializing in the technical and non-technical aspects of software reusability. He is a member of IBM Corporate Reuse Council and IBM FSD Reuse Steering Committee as well as an editor for the IBM Corporate Programming Reuse Newsletter. Tracz has written over 25 publications on software reuse, programming languages, and microprogramming. He has given keynote addresses as several workshops and conferences on software reuse. His book, *Software Reuse: Emerging Technology*, published by IEEE Computer Society Press, 1988, paints a broad picture of the technical, economic, pedagogical and social issues facing the transfer of software reuse technology into the work place. Currently, Tracz is co-principle investigator for the DARPA Domain Specific Software Architecture project focusing on the Avionics application domain.

Lou Coglianese

Lou Coglianese is a senior programmer at the Owego Laboratory of the IBM Federal Sector Division specializing in large-scale software reuse as it applies to Avionics and ESM real-time software development. He was the designer and developer of the Reusable Avionics Ada Software Proof-of-concept program as well as the developer of Owego Software Component Guidelines for Domain Analysis and Large Scale Reusable Software. Coglianese is the former chair of the Owego Reusable Software Advisory Board. Currently he is co-principle investigator for the DARPA Domain Specific Software Architecture project focusing on the Avionics application domain.