A DESIGN PATTERN BASED TECNIQUE FOR RUNTIME OBSERVATION, MEASUREMENT AND REPORTING

Radonjic Vojislav D.[1], Bashardoust Tajali Soheila[1], Arnold Dave[1], Corriveau Jean-Pierre[1], and Mihajlovic Radomir A.[2]
[1]Carleton University, Ottawa, Canada, radonjic@acm.org
[2]NYIT, New York, NY, USA, rmihajlo@nyit.edu

***Abstract**: This paper describes a technique based on the widely accepted concept of design patterns, with a twist: rather than it's conventional use as a forward engineering tool, it is used in the opposite direction, as a runtime observation, measurement and reporting tool.*

***Keywords**: Software engineering, model driven engineering, design pattern, run time observation, design variant selection.*

1. INTRODUCTION

Modern software engineering promotes modeling of requirements and corresponding design, and, to a limited extent, traceability of requirements into design and implementation. Broadly, the techniques and tools associated with development and use of models are called Model-Driven Engineering (MDE) or Model-Driven Development (MDD). We can think of software designers as bridge builders from the problem world of requirements to the solution world of various candidate systems. Design patterns have emerged as important tools for documenting and sharing the 'bridge building' experience of spanning from the side of requirements (scenarios and features) to the side of implementations (C++, Java, various run-times). Like craftsmen in other disciplines, software designers place great importance on their tools, and judging by the large number of copies of the GoF catalog [3] sold, and by the ubiquity of these patterns in newly developed systems, these craftsmen regard design patterns as one of their essential problem solving tools.
Design patterns, however, remain a challenge for the modeling community. The core pattern conceptualization – the often repeated 'a design solution to a problem in context' – is considerably more complex. We go in some detail to illustrate that in [1] in the example of the Factory Method pattern, and more broadly the creational family of patterns.

Beyond reuse and comprehension in early software development phases, the question is can design patterns be helpful in software quality assurance, for individual systems, a family of systems, or across generations of related system? Our answer is yes, assuming we have 1) a model that is precise enough to be a basis for observation and measurement, and 2) a corresponding tool to carry out the observation and measurement and generate evaluation reports.

2. CHALLENGES MODELING DESIGN PATTERNS

Design Patterns and their use are challenging to represent with existing modeling techniques [1]. In particular, it is not clear how we can evaluate a decision made in selecting a given pattern and a given variant of that pattern. The first step is making visible the space of choices and the criteria for selection by modeling the content of a pattern description.
Helm et al. [5] present the 'root' modeling attempt at representing reusable and implementation-independent object-oriented designs. They chose precision over ease of use of the model by the target audience, the day to day developer: the result was few users. A few years on, with the lesson learnt, a more accessible and informal, template based model was used to represent the GoF design pattern catalogue [3]: the biggest selling software engineering book to date [9]. Our approach, in essence, aims to achieve the precision of the contract model, with the ease of use of the GoF model.
Design Patterns are complex abstractions that span from requirements through design to implementation, with variability at each phase, their use is challenging to model because of the following:

- Traceability [12] from use cases to variations and their implementations,
- Variability modeling [13] in analysis, design, and implementation,
- Interaction between traceability and variability, and
- Evaluation of selected variants relative to intents and detailed design properties, and corresponding implementations

The model we have developed [1] to address the above challenges is based on generative modeling and programming techniques [8], combined with modeling-by-contract [2][4][5][6].

3. IMPLEMENTING THE TECHNIQUE

We outline an implementation of the technique in ACL (Another Contract Language), VF (Validation Framework), and the variability resolution framework, Figure 1 and Figure 2 respectively. In our presentation, we will provide a walkthrough of our approach. In particular, we will show how the selection forces are represented through metrics, and how the implementations are evaluated both in terms of their functional and non-functional design requirements.
Use-cases [11] are a well-established, understood and standard technique for requirements modeling, however, they lack precision necessary for use in executable forms of system evaluation. Corriveau and Arnold have addressed this in ACL [2], an executable requirements modeling language. Arnold has developed a corresponding tool, Validation

Framework (VF) for evaluating .NET executables relative to ACL specified requirements models [2].

Figure 1: Validation Framework [2].

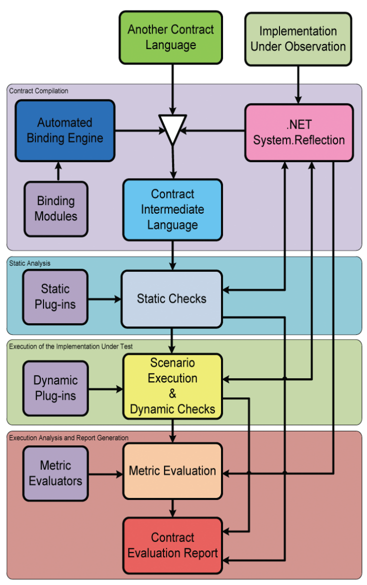
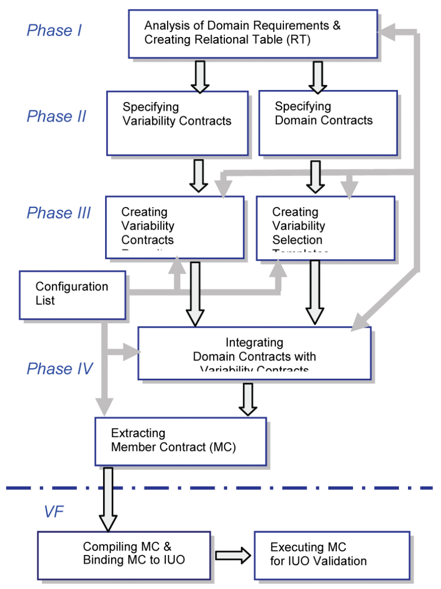

Another Contract Language
Implementation Under Observation
Contract Compilation
Automated Binding Engine
.NET System.Reflection
Binding Modules
Contract Intermediate Language
Static Analysis
Static Plug-ins
Static Checks
Execution of the Implementation Under Test
Dynamic Plug-ins
Scenario Execution & Dynamic Checks
Execution Analysis and Report Generation
Metric Evaluators
Metric Evaluation
Contract Evaluation Report

Figure 2: Generative Framework [16].


Phase I
Analysis of Domain Requirements & Creating Relational Table (RT)
Phase II
Specifying Variability Contracts
Specifying Domain Contracts
Phase III
Creating Variability Contracts
Creating Variability Selection
Configuration List
Phase IV
Integrating Domain Contracts with
Extracting Member Contract (MC)
VF
Compiling MC & Binding MC to IUO
Executing MC for IUO Validation

The Validation Framework, Figure 1, allows us to evaluate candidate implementations of ACL specified design patterns and their variants. In particular, it allows us to implement a model of traceability from designs to implementations; specifically, from detailed design properties, i.e. consequences, to candidate implementations.
Modeling and resolution of variability is handled through a combination of generative modeling and contract-based techniques. The framework under development, Figure 2, is discussed in detail by Bashardoust Tajali in [16]

5. CONCLUDING REMARKS

In this work we outline our novel approach to design pattern based technique for observation, measurement and reporting that we envision can be used for evaluation of implementations relative to selected design pattern variants, intents and detailed design properties.
Particular challenges are:

- Variability
  The complexity of modeling even a few variants of a straight-forward mediator pattern is high, let alone the ability to use that model to evaluate implementations conformance. The challenge is unavoidable because dealing with variability is essential to handling complexity in software systems.

- Navigation and Selection
  The variants found in each pattern are points in some design subspace: the 8 variants of the Factory Method pattern define a Factory Method subspace in the larger Creational pattern design space. [1] We can imagine the designer as a navigator in that space, moving towards one pattern subspace over another under the influence of forces, and selecting a particular point, i.e. variant, in that space. In these terms, the GoF format provides a first-level, sparse description of both the space defined by a particular pattern, and the forces acting upon a designer to select a pattern.
- Evaluating Choices
  How do we evaluate that the chosen variant satisfies the specific goal and that the implementation satisfies the properties of the design?

REFERENCE

[1] V.D. Radonjic, S. Bashardoust, J.-P. Corriveau and D. Arnold, Design Patterns – A Modeling Challenge, Proceedings of SERP'11, Las Vegas, Nevada , USA, 2011. (pp191-197) available at http://world-comp.org/proc2011/serp/papers.pdf
[2] D. Arnold and J.-P. Corriveau, Validation Framework and Another Contract Language, http://vf.davearnold.ca/
[3] E. Gamma, R. Helm, R. Johnson, & J. Vlissides, Design Patterns-Elements of Reusable Object-Oriented Software (Reading, MA: Addison-Wesley, 1994).
[4] B. Meyer, "Design by Contract," IEEE Computer, vol. 25, no. 10, pp. 40-51, October 1992.
[5] Helm, R., Holland, I., Gangopadhyay, D.: Contracts: Specifying Behavioral Compositions in Object-Oriented Systems. In proceedings of the Object-Oriented Programming Systems, Languages and Applications Conference (OOPSLA'90), pp. 169-180, October 1990.
[6] J.-M. Jezequel, M. Train, and C. Mingins, Design Patterns and Contracts, Addison-Wesley, 2000.
[7] L. Ackerman and C. Gonzalez, Pattern-Based Engineering, Successfully Delivering Solutions via Patterns, Addison-Wesley, 2011.
[8] K. Czarnecki and U. Eisenecker: Generative Programming: Methods, Tools, and Applications, Addison-Wesley, June 2000.
[9] http://en.wikipedia.org/wiki/Design_Patterns
[10] M. Antkiewicz, K. Czarnecki, and M. Stephan, Engineering of Framework-Specific Modeling Languages, IEEE Trans. Software Eng., vol. 35, no. 6, pp. 795-824, Nov/Dec 2009.
[11] I. Jacobson, M. Christerson, P. Jonsson and G. Overgaard: Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992.
[12] J.-P. Corriveau, Traceability Process for Large OO Projects, IEEE Computer, pp. 63-68, Sep 1996.
[13] S. Bashardoust, V.D. Radonjic and J.-P. Corriveau and D. Arnold, Challenges of Variability in Model-Driven and Transformational Approaches: A Systematic Survey, Workshop on Variability in Software Architecture, WICSA2011, Boulder, Colorado, USA, 2011.

[14] D. Arnold, Grocery Store (available at http://designpatterns.elmdale.ca/TheGroceryStoreExample_V3_1.pdf)
[15] S. Bashardoust, Grocery Variability (available at: http://designpatterns.elmdale.ca/SoheilaGroceryVariability.zip)
[16] S. Bashardoust, Generative Contracts, Doctoral Dissertation, School of Computer Science, Carleton University, Nov 2012.