Chapter 12

Model–Driven Exception Management Case Study

Susan Entwisle
Monash University, Australia

Sita Ramakrishnan
Monash University, Australia

Elizabeth Kendall
Melbourne Institute of Technology, Australia

ABSTRACT

Programming languages provide exception handling mechanisms to structure fault tolerant activities within software systems. However, the use of exceptions at this low level of abstraction can be error-prone and complex, potentially leading to new programming errors. To address this we have developed a model-driven exception management framework (DOVE). This approach is a key enabler to support global distributed solution delivery teams. The focus of this paper is the evaluation of the feasibility of this approach through a case study, known as Project Tracker. The case study is used to demonstrate the feasibility and to perform an assessment based on quality and productivity metrics and testing of the DOVE framework. The results of the case study are presented to demonstrate the feasibility of our approach.

INTRODUCTION

Rapid advances in information technology and consumer demand have motivated the development of increasingly rich, complex software systems. Within these software systems, exceptions are inevitable and increasing due to the diverse range of potential points of deviations and errors, including hardware, software, and human error (Garcia, F.Rubira, Romanovsky, & Xu, 2001; Patterson et al., 2002). Modern programming languages, including Java (Arnold, Gosling, & Homes, 2000) and C# (Williams, 2002), provide exception handling mechanisms to systematically handle errors within these software system to meet reliability requirements (Alexander Romanovsky, 2003; Perry, Romanovsky, & Tripathi, 2000). However, while exception handling can improve the design of an
application, the use of exception handling is labor intensive, error prone, provides limited support for communication, and is inflexible (Howell & Vecellio, 2000; Klein & Dellarocas, 2000; Reimer & Srinivasan, 2003).

Much of the exception handling research to date has concentrated on addressing the objectives of exception handling as a language design issue (Garcia et al., 2001), or providing the software developer with exception usage design patterns and software frameworks (Anderson, Feng, Riddle, & Romanovsky, 2003; Garcia & Rubira, 2000; Reimer & Srinivasan, 2003). However, despite the advantages, experience has shown that progress in understandability and reusability within most problem domains other than limited vertical frameworks and class libraries, (such as user interface, databases, virtual machines, etc), has failed to meet expectation (Eisenecker & Czarnecki, 2000; Greenfield, Short, Cook, & Kent, 2004).

This finding extends beyond just exception handling, to a wider range of software development. (Coplien, 1998; Eisenecker & Czarnecki, 2000; Greenfield et al., 2004; Mellor, Scott, Uhli, & Weise, 2004) argue that significant advances in reuse and the management of complexity can be potentially achieved by transitioning to system families. These families are represented using higher level abstractions of the problem domain, represented as models, in order to meet problem domain requirements. Model transformation facilitates the automatic construction of highly customized intermediate or end applications based on high-level problem domain models. These models are configured to leverage elementary, reusable domain-specific assets rather than programming languages (Eisenecker & Czarnecki, 2000; Greenfield et al., 2004; Mellor et al., 2004). The model-driven approach is a key enabler to support global application delivery teams to design and deliver projects using geographically distributed teams. This approach allows the architect that is co-located with the customer to design the system using high-level models. These models can then be provided to the application development team to automatically construct the application.

This approach aims to promote productivity, maintainability, and expressiveness, and aid in the management of complexity by supporting higher levels of abstraction that systematically reuse domain-specific assets (such as domain-specific languages, common architecture, components, documentation, etc). To achieve this, model-driven development is based on software product-line engineering practices (also known as system-family engineering). This approach seeks to capture commonalities among systems within a given problem domain while managing variations in a systematic manner (Paul Clements & Northrop, 2001; Eisenecker & Czarnecki, 2000; Greenfield et al., 2004; Weiss & Lai, 1999). This family based software development process is divided into two parts: domain engineering that focuses on engineering-for-reuse and application engineering that focuses on engineering-with-reuse.

The limitations of language-based approaches to exception management and the potential benefits of model-driven development have motivated our research into model-driven development for exception management (Entwisle, 2007). Researchers such as Capozucca et al. and Greenfield et al. (Capozucca, Gallina, Guelfi, Pelliccione, & Romanovsky, 2006; Greenfield et al., 2004) have expressed the importance of providing support for non-functional requirements, such as security, exception handling, and so on, within model-driven approaches in order to manage complexity, promote communication amongst stakeholders, and improve software quality. As discussed in Czarnecki et al. and Greenfield et al. (Czarnecki, Eisenecker, Gluck, Vandevoorde, & Veldhuizen, 2000; Greenfield et al., 2004), the model-driven development approach provides the necessary processes and techniques to leverage high quality domain-specific assets, subsequently leading to significant benefits in the development of software systems. To date, the application of model-driven
19 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the product’s webpage: www.igi-global.com/chapter/model-driven-exception-management-case/37031?camid=4v1


Related Content

Watermarking Scheme with CS Encryption for Security and Piracy of Digital Audio Signals

Software Service Adaptation Based on Interface Localisation
www.igi-global.com/article/software-service-adaptation-based-on-interface-localisation/125842?camid=4v1a

Software Engineering at Full Scale: A Unique Curriculum
www.igi-global.com/chapter/software-engineering-full-scale/29603?camid=4v1a

ETCS Developing and Operation: Italian Experience
www.igi-global.com/chapter/etcs-developing-operation/66682?camid=4v1a