

Contents

	Page
Editorial	5

COLUMNS

Strategic Software Engineering

Ecosystems, continued	7
<i>By John McGregor</i>	

At Philips, the very large size of the company makes it possible to think of the product line organization as an ecosystem within the corporation. The inner source approach used by Philips establishes an ecosystem that involves several business units in the corporation. The ecosystem extends outside the company to include the users of Philips equipment, a highly skilled audience, and more indirectly organizations that learn from the experience of Philips regarding the open source techniques.

Java at Large

The Discrete Fourier Transform, Part 4: Spectral Leakage	23
<i>By Douglas Lyon</i>	

Windowing modulates the input signal so that the spectral leakage is evened out (spreading on-bucket signals more and off- bucket signals less). Thus, windowing reduces the amplitude of the samples at the beginning and end of the window, altering leakage.

Business Objects

Cloud Architecture	35
<i>By Mahesh Dodani</i>	

The cloud architecture needs to provide the support for different types of workloads, especially related to their qualities of service (QoS) requirements, the different types of images that are needed to deliver the types of services implied by the workloads, and any special business and compliance policies that they need to adhere to.

Guest column**Types and Co-Types in Timor**

37

By Leslie Keedy, Gisela Menger, and Christian Heinlein

About forty years ago Doug McIlroy wrote a widely quoted paper in which he advocated that software should be built in terms of small units which can have many different implementations. He envisaged that in this way software companies could compete in providing small components and that these components could be built into many different systems. This vision (which in our view differs substantially from that of those who see components as larger structures) is shared by the designers of Timor, and we see the way forward as being realized in an object oriented style, with the help of the information hiding principle, proposed at about the same time by David Parnas.

REFEREED ARTICLES

Exploring the use of Package Templates for flexible re-use of Collections of related Classes

59

By Stein Krogdahl, Birger Møller-Pedersen, and Fredrik Sørensen

Concepts of some complexity are often most naturally implemented by more than one class. A typical example is the concept of a graph, which normally will have classes for nodes and edges. Thus, it has for some time been recognized that there is a need for a language mechanism that support re-use of collections of related classes. And, because we want the collections to be as generally useful as possible, we want a mechanism that allows the classes of a collection to be tailored to specific use situations.

A Meta-Model for Textual Use Case Description

87

By Stéphane S. Somé

The meta-model described in this paper has been used as basis for two use case modeling tools. The first tool is an Eclipse plugin developed using the Eclipse Modeling Framework (EMF) and the Eclipse Model Development Tools (MDT). EMF automates the generation of editors from models while the MDT includes a reusable EMF-based implementation of the UML meta-model as well as an implementation of the OCL for EMF models. The resulting tool is a very basic use case editor with little usability. However, this implementation allowed us to connect our meta-model to the UML meta-model and validate the OCL constraints.



First-Class Connectors to Support Systematic Construction of Hierarchical Software Architecture 107

By Abdelkrim Amirat and Mourad Oussalah

Having a representation of software architecture allows an easy exchange between the architect and programmer. Also, during the phases of maintenance and evolution, this representation helps to locate defects and reduces the risk of improper assembly of a new feature in the system. In addition, the distinction which exists between components and connectors allows a more explicit representation between the functional aspects and these of communication and therefore, makes the system easier to understand and to change.

Components, Contracts and Vocabularies - Making Dynamic Component Assemblies more Predictable 131

By Jens Dietrich, and Graham Jenson

In recent years, dynamic component-based systems such as OSGi and its derivatives have become very successful. This has created new challenges for verification. Assemblies are created and modified dynamically at runtime, but many existing techniques such as unit testing are designed for buildtime verification. Runtime verification is usually restricted to type checks. We propose a simple component contract language that is powerful enough to represent different types of complex contracts between collaborating components, including contracts with respect to component semantics and quality of service attributes, and contracts that refer to resources other than programming language artefacts.

Method Proxy-Based AOP in Scala 149

By Daniel Spiewak and Tian Zhao

This paper describes a framework to implement an AOP extension to the Scala language using higher-order functions as AOP proxies. This framework allows programmers to specify pointcuts and aspects using a Domain Specific Language (DSL) embedded within Scala. The technique uses Scala's higher-order functions to intercept method calls with minimal syntactic overhead imposed on the base program.

OUTLOOK**A brief outlook to the next issue** 171