

eclipse

POWERING THE

MAGAZINE

ECLIPSE ECOSYSTEM

Dynamic Wizard Modeling with GMF

Using GMF to Build a Dynamic Wizard Framework and a Graphical Editor

Introduction to the

Generic Eclipse Modeling System

Developing a Graphical Modeling Tool for Eclipse

Deploying the BIRT Viewer to JBoss

Disseminate Report Content to an Application Server

Subversive

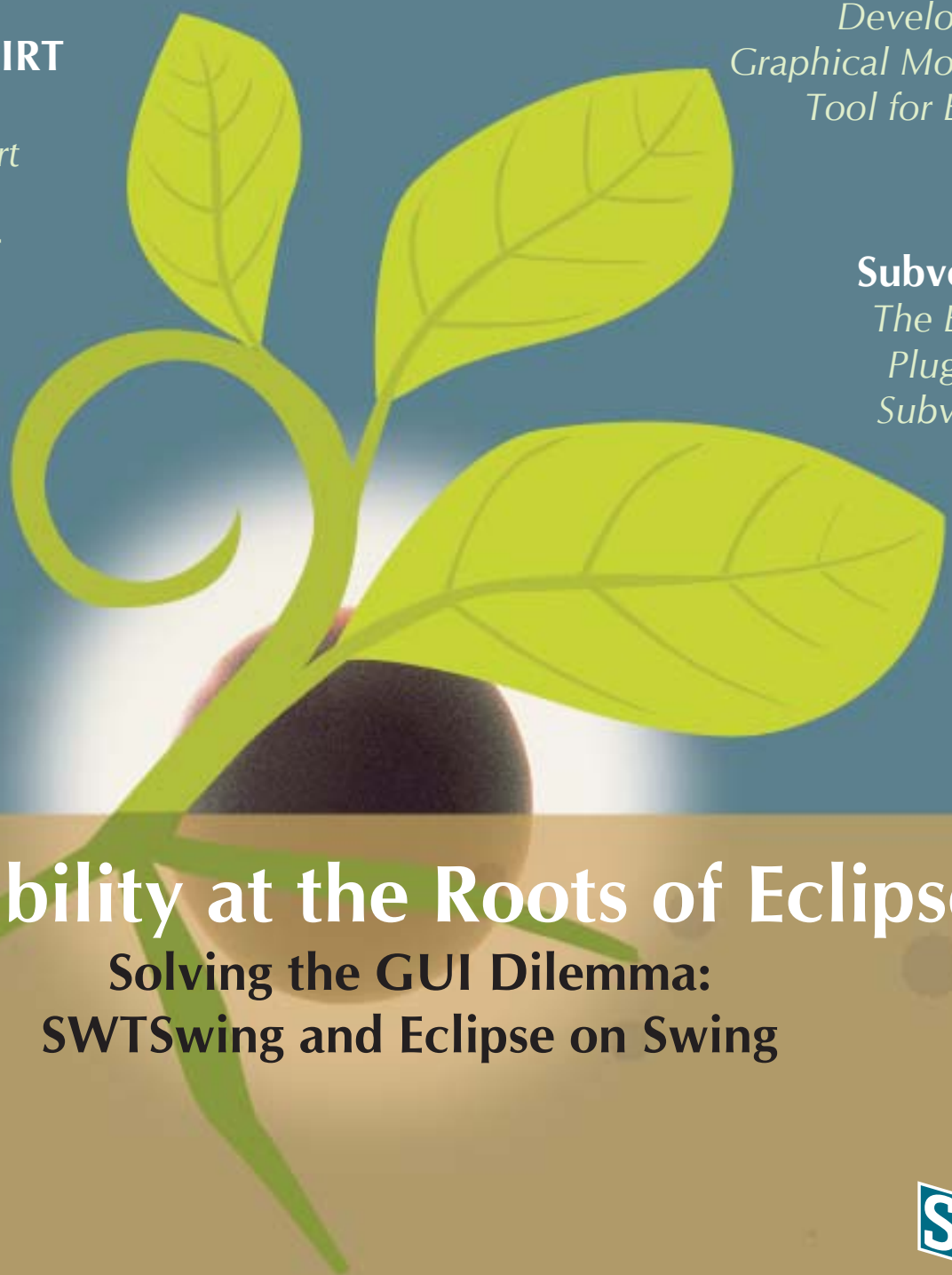
The Eclipse Plug-In for Subversion

Enabling Integration and Interoperability for Eclipse based Development

An Introduction to the Corona Project

Flexibility at the Roots of Eclipse

**Solving the GUI Dilemma:
SWT Swing and Eclipse on Swing**



FEATURES

29 Flexibility at the Roots of Eclipse

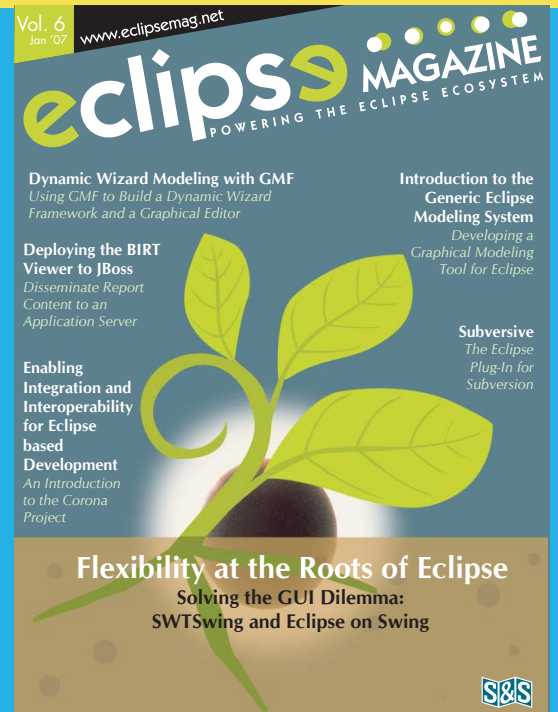
Solving the GUI Dilemma: SWTswing and Eclipse on Swing

No trench in the world of Java is deeper than that between SWT and Swing or Eclipse and Sun. Unity is only found in the knowledge that everybody suffers from this argument. But how to end this almost religious battle over the righteous GUI-toolkit? How to bang their heads together if they only know one point of view—for them or against them! The sister projects SWTswing and Eclipse on Swing (EOS) achieve this trick. They offer both wranglers a conciliative solution, which combines the best of both worlds and seemingly has only advantages for everybody.

By Dieter Krachtus and Christopher Deckers

11 Introduction to the Generic Eclipse Modeling System

Developing a Graphical Modeling Tool for Eclipse
Graphical Model-Driven Engineering (MDE) tools have become extremely popular in the development of applications for a large number of domains. In many cases, however, an organization does not have the resources or time available to develop a graphical modeling environment from scratch using the Eclipse Modeling Framework (EMF), Graphical Editor Framework (GEF), or Graphical Modeling Framework (GMF). In other situations, the complexity of the domain limits the feasibility of using a graphical model to describe a domain solution. The Generic Eclipse Modeling System (GEMS), which is part of the Eclipse Generative Modeling Technologies (GMT) project, helps developers rapidly create a graphical modeling tool from a visual language description or metamodel without any coding



DEPARTMENT

05 News & Trends

Reporting the latest announcements from the community, Tracking new releases of developmental tools.

FEATURES

44 Enabling Integration and Interoperability for Eclipse-based Development

An Introduction to the Corona Project
Designing, developing, testing and managing business critical applications has become increasingly more complex. To manage this complexity, projects are typically divided into tasks and teams are assigned to execute them. But IT managers also need to ensure all these different teams and team members collaborate effectively as if they were a small team all working in the same room, in the same office, and on the same project. The article explains why Corona is the right tool to address this IT business problem.

By Edwin Shumacher

in third-generation languages. GEMS automatically generates the requisite EMF, GEF, and GMF code required to implement the editor from a metamodel. GEMS also provides extensive capabilities for expressing modeling guidance and performing optimization. Finally, graphical modeling tools created with GEMS automatically support complex capabilities, such as remote updating and querying, template creation, styling with Cascading Style Sheets (CSS), and model linking.

By Jules White, Douglas C. Schmidt, Andrey Nechypurenko, Egon Wuchner

20 Dynamic Wizard Modeling with GMF

Using GMF to Build a Dynamic Wizard Framework and a Graphical Editor

Developing a graphical editor is generally very complicated and requires lot of effort. There are few frameworks available for writing graphical editors in Java. The prominent open source frameworks are JHotDraw (which is Swing based) and GEF (which is SWT/Jface-based). While they provide sophisticated tools for graphical development, the painstaking work of modeling the domain and mapping to graphical elements is left to the user. Graphical Modeling Framework (GMF) bridges this gap nicely. In the article, I will take you through an end-to-end demonstration of GMF. To achieve that, first we will create a framework for meta-data driven JFace wizards. Next, we will see how to use GMF to build a graphical editor for this framework.

By Rajkumar C Madhuram

38 Deploying the BIRT Viewer to JBoss

Disseminate Report Content to an Application Server

With information applications, developing content for delivery is only a part of the equation. After report designs are complete, the infrastructure for deploying the application has to be addressed. This is equally true for BIRT applications. In previous articles, we discussed many of the BIRT Designer's features, but in this one we will cover deployment options available to the BIRT developer, with an emphasis on deploying the Example BIRT Viewer to the JBoss Application Server—although the techniques discussed in this article should apply to most J2EE-compliant application servers. core capability required by agile developers.

By Jason Weathersby

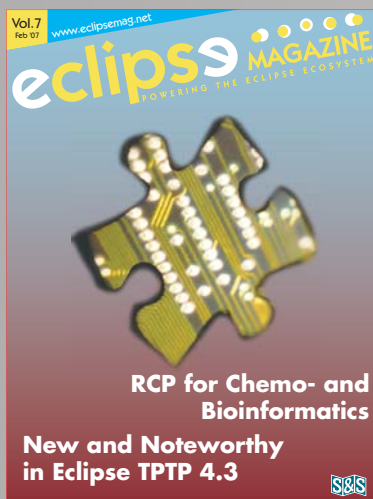
35 Subversive

The Eclipse Plug-In for Subversion

Version control systems play a central role in software engineering. In the beginning, CVS was the poster child versioning system of the open source community. Then Subversion was developed because many felt CVS could not keep pace with changing technologies and practices. New plugins such as Subversive and Subclipse have appeared in the Eclipse ecosystem to connect Eclipse developers and Subversion. The article introduces you to Subversive.

By Frank Schröder

PREVIEW



Eclipse Forum brings together Eclipse experts and enthusiasts from all over Europe for a meeting that is guaranteed to deliver profound know-how on the Eclipse platform and presents on top of this excellent networking opportunities for all participants.

eclipse FORUM 2007

Eclipse Forum Europe
Date: 23-27 April, 2007
Venue: Rhein-Main-Hallen,
Wiesbaden, Germany

Eclipse Forum India
Date: 28-31 May, 2007
Location: Indian Institute
of Science (IISc),
Bangalore

Eclipse Forum Asia
Date: 26-29 November,
2007
Location: Singapore

New Tidings for the New Year

Greetings for the year 2007 from all of us here at Eclipse Magazine. We have a lot of interesting things lined up for you this year. Apart from ensuring that you get round-the-clock information about Eclipse and the developments in the community through this magazine and <http://www.eclipsemag.net/>, we will also be presenting the Eclipse Forum conference at four locations in Europe and Asia this year. The first conference, www.eclipseforumeurope.com, will be held from April 23-27 in Frankfurt, Germany. Next, we move to India to present the conference, www.eclipseforumindia.com, to the Eclipse community in the Indian subcontinent, by presenting a three-day extravaganza in Bangalore, India from May 28-31. We'll end the year with the conferences (jax-asia.com) in Singapore and Indonesia from 26-29 November.

Let's now move on to what's in store for you this issue. In the Cover story, Dieter Krachtus and Christopher Deckers throw light on two projects that are attempting to end the almost religious battle over the righteous GUI-toolkit. The choice of a toolkit is often based on the taste of the developer, which means there was no real choice from the beginning. But what if you had the facility to choose between SWT and Swing not just at the beginning of the project, but throughout your development phase? How about using familiar APIs to develop Eclipse-Plug-ins, RCP-applications or a JFace/SWT GUI and still keep the option to switch back and forth between SWT and Swing without changes to your code? SWT/Swing offers this possibility, and Eclipse on Swing (EOS) is proof that it even works for the most complex SWT applications, namely Eclipse itself.

Organizations sometimes don't have the luxury of developing a graphical modeling environment from scratch using the Eclipse Modeling Framework, Graphical Editor Framework, or Graphical Modeling Framework. Jules White, Douglas C. Schmidt, Andrey Nechypurenko, and Egon Wuchner explain how the Generic Eclipse Modeling System, which is part of the Eclipse Generative Modeling Technologies (GMT) project, helps developers rapidly create a graphical modeling tool from a visual language description or metamodel without any coding in third-generation languages.

While the Swing-based JHotDraw and the SWT/Jface-based GEF provide sophisticated tools for graphical development, the painstaking work of modeling the domain and mapping


to graphical elements is left to the user. Graphical Modeling Framework (GMF) bridges this gap nicely. Raj Madhuram's article steps you through the process of creating a dynamic wizard framework and a graphical editor using GMF from scratch.

Version control systems play a central role in software engineering. Earlier, CVS was the poster child versioning system of the open source community. Subversion was developed because many felt CVS could not keep pace with changing technologies and practices. New plugins such as Subversive and Subclipse have appeared in the Eclipse ecosystem to connect Eclipse developers and Subversion. In Plug-in Parade, Frank Schröder takes you on a journey through Subversive.

Last year, we had a lot of articles focused on BIRT Designer, and based on popular demand you will see many more in the forthcoming issues this year. The article in this issue goes further to tackle deployment options available to the BIRT developer. Developing content for delivery is only a part of the equation. After report designs are complete, the infrastructure for deploying the application has to be addressed. Jason Weathersby steps you through deploying the Example BIRT Viewer to a JBoss Application Server. The techniques discussed in this article are applicable to most J2EE-compliant application servers.

Designing, developing, testing and managing business critical applications has become increasingly more complex. Additionally, to ensure applications are developed efficiently and delivered on time, projects are typically divided into tasks and teams are assigned to execute them. At the same time, IT managers also need to ensure all these different teams and team members collaborate effectively as if they were a small team all working in the same room, in the same office, and on the same project. Edwin Schumacher explains why Corona is the right tool to address this IT business problem.

We hope you enjoy what we have lined up for you this issue. Until next month, *Happy New Year* from the Eclipse Magazine team.



Indu Britto

Editor-in-Chief, Eclipse Magazine

— Announcements —

Eclipse Challenges Visual Studio.NET



The adoption rate of Eclipse has more than doubled in the EMEA region, making it the first IDE to challenge Microsoft Visual Studio.NET among developers there, according to the recently released Evans Data

EMEA Development Survey. The study shows that one in every three developers in the region now uses Eclipse and the use of the set of plug-ins needed to develop rich-client applications (Eclipse RCP subset), is expected to triple within the next two years.

A related survey of the APAC has also shown similar results—it was found that Eclipse usage in the region has grown by 80 per cent in the last six months, and it is now the most popular IDE in the region except for Microsoft's Visual Studio

“These results are reflective of the overall market adoption we are seeing for Eclipse adoption globally,” John Andrews, President, Evans Data added. “The addition of the Rich Client Platform provides the developers a great advantage being able to write an application once and it will run on Windows, Mac OS and Linux.”

Other findings from the Winter 2006 survey of almost 400 developers in the EMEA region:

- Forty-two percent of EMEA developers use XML as a component of their development efforts
- AJAX demonstrates the highest growth rate (more than 12 per cent) amongst web service standards, consistent with XML expansion
- Java frameworks are considered the most important Java feature, although refactoring emphasis more than doubled during the period, suggesting added focus on coding efficiency associated with Java development efforts

[\[MORE INFO\]](#)

Google Takes its First Steps in the Eclipse World



Google has joined the Eclipse Foundation as an add-in provider. But that really is old news. Google joined the Eclipse Foundation way back in October, but they decided not to let

the world until now. The Eclipse Foundation normally does not announce memberships. It is also not uncommon for members to either not announce, or delay, the news of their membership. The news about Google's membership was made known at the Quarterly Eclipse Membership meeting where members are given an opportunity to present themselves and speak about what they are doing with Eclipse.

With Google joining Eclipse, the number of large companies that aren't yet part of the Eclipse movement are becoming far and few. Mike Lilinkovich, Director of the Eclipse Foundation said that, “We are, of course, thrilled that Google decided to join the Eclipse Foundation. It was a nice gesture of support for the Eclipse community and it is appreciated. But what I am really excited about is the way that Google is using Eclipse as the development tools platform for the Google Web Toolkit. It is yet another demonstration of how Eclipse can be used as the tools platform for so many different languages and environments.”

Google has also open sourced the Google Web Toolkit, which is built on Eclipse RCP and JDT technology. GWT 1.3 Release Candidate is 100 per cent Open Source, and is released under the Apache 2.0 license. “Now that GWT has some serious adoption and a lively user community, open-sourcing is the obvious next step to help GWT evolve more quickly,” says the project's Tech Lead, Bruce Johnson, in the GWT blog. What is worth noticing though is that this decision came just after the news of Google joining the Eclipse Foundation, as an add-in provider became public.

So what is new in 1.3 RC1? “Well, nothing, actually. The only thing that has intentionally changed since GWT 1.2 is the open source thing. However, since we did have to tweak a lot of source code and the GWT build scripts to make it easy for anyone to compile, we want to be conservative and call this a Release Candidate anyway. Once we hear enough success stories with the 1.3 RC, we'll call it 1.3 Final then get back to full-speed-ahead coding as we drive toward a nice big GWT 1.4.”

[\[MORE INFO\]](#)

Open Source Intellectual Property Compliance Solution for Eclipse



For organisations using open source, it is critical that a deliberate and methodical process for managing intellectual property compliance is part of the software development process.

Palamida, a provider of software assurance management products and audit services, has announced a partnership with Innoopract, a provider of Eclipse-based add-on tools, consulting, support, training and open source distribution solutions. The partnership aims to provide organisations seeking to adopt open source while respecting intellectual property rights, a combined Innoopract/Palamida solution to ensure they are meeting compliance requirements.

The aim is to accelerate the adoption of open source in the enterprise, with their combined technology and service offerings, the companies claim. Palamida and its partners perform software code audits using Palamida's IP Amplifier technology and the compliance library of third-party software, to accurately assess the origin of each client's software code. The software code audit report is touted to provide the assessment of open source and third-party code, pinpointing specific high-risk areas of company code against due diligence criteria to identify remediation issues. Organisations can use the companies' combined services to be fully prepared to extend their Eclipse development environment without violating copyright or intellectual property rights.

"Open source adoption is accelerating at an incredible rate this year," said Susanna Kass, Executive Vice President and COO of Palamida. "With Innoopract's outstanding track record in Eclipse project distribution and Ajax development tools, this strategic partnership can help enterprise software and system vendors, as well as corporate users, achieve software transparency and effectively and safely embrace the benefits of open source software development."

[\[MORE INFO\]](#)

TRANGO Systems Joins The Eclipse Foundation



TRANGO Systems has joined the Eclipse Foundation as an Add-In Provider. Specializing in embedded hardware virtualization, TRANGO Systems helps semiconductors and devices manufacturers to build up scalable and secure platform. By joining Eclipse Foundation TRANGO Systems will contribute to the open source IDE growing adoption among embedded developers community. Since April 2006, TRANGO Systems products are shipped with Eclipse plug-ins for system integration, target control, monitoring and virtual-JTAG debugging.

The TRANGO hypervisor allows multiple execution environments to run securely side by side on the same processor core, which, as a result, enables to securely deploy new services and reduce Bill of Material. By offering a safe environment for secure deployment of new services such as DRM or Device Management, TRANGO Systems helps operators and content providers growing new revenues. The TRANGO Hypervisor also allows systems to be certified and thus helps compliancy with security standards and requirements.

"We warmly welcome TRANGO Systems as a member of the Eclipse Foundation. As the company is working with worldwide leading semiconductors manufacturers, we expect that the TRANGO Hypervisor adoption will help strengthening Eclipse IDE use within semiconductors and devices industry" stated Mike Milinkovich, Executive Director of the Eclipse Foundation. Available on ARMv5 and MIPS32 / MIPS64 embedded architectures, the TRANGO Hypervisor and its Eclipse tools support Linux, Windows CE 5.0, as well as eCos and μ C/OS-II. [\[MORE INFO\]](#)

Terra Opts for Open Source BI



Pentaho, a provider of Business Intelligence (BI) capabilities, has announced that international nitrogen products producer Terra Industries has selected the Pentaho BI platform to provide advanced reporting and analysis based on data in Terra's SAP R/3 system. Terra selected Pentaho after evaluating multiple proprietary and open source BI offerings. "We chose Pentaho because it has a full range of functionality, exceptional flexibility, and a low total cost of ownership because of its open source business model," said James Keairns, Terra's Director, Information Technology.

Initially, Terra business users who manage production and inventory will use Pentaho Reporting to integrate customer demand and production volumes as part of a planned web application that will be built using Eclipse IDE. Over the longer term, the company also plans to use Pentaho Data Integration to extract and integrate operational information from SAP R/3, Peoplesoft HR, Tabware, and other systems to provide a centralised view. According to Lance Walter, VP of Marketing for Pentaho, this development signifies a clear

need in the marketplace for an open source alternative to proprietary BI systems, even for large organizations.

[\[MORE INFO\]](#)

IBM Donates Translations for Eclipse 3.2.1



IBM has announced the contribution of translations for the Eclipse Project, the Eclipse Web Tools Platform (WTP) Project, the Eclipse Test and Performance Tools Platform (TPTP) Project, the Business Intelligence and Reporting Tools (BIRT) Project, the Eclipse Modeling Project, the Eclipse Data Tools Platform (DTP) Project and for several subprojects of the Eclipse Tools Project for the Callisto releases.

The language packs are distributed as zips, which can be installed by downloading the zip file and unzipping into an Eclipse directory before starting Eclipse. These zips appear on each project's download pages.

Each NL feature description contains a list of the language translations provided by that NL feature. NL features are new features that did not exist for the component prior to translation. NL features parallel the set of features defined by the component and contain the NL fragments that in turn contain the translations for the component's plugins.

[\[MORE INFO\]](#)

EclipseCon 2007 Keynotes by Dilbert Creator, Robert Lefkowitz, Herbert Thompson



The Eclipse Foundation has announced the keynote speakers for its fourth annual community conference, EclipseCon 2007. The conference brings together the

Eclipse community to share and learn new techniques, ideas and technologies, as well as provide information about the latest developments and future plans of the Eclipse projects. The conference will be held March 5-8, 2007 at the Santa Clara Convention Center.

The conference will feature 50 in-depth tutorials, 65 long-talk sessions, 100 short-talk sessions and 16 panels. The keynote speakers for EclipseCon 2007 will include Scott Adams, creator of the comic strip Dilbert; Robert "r0ml" Lefkowitz, an enterprise architect and frequent speaker on open source software, and Dr. Herbert Thompson, expert on

application security and Chief Security Strategist for Security Innovation.

EclipseCon 2007 will also host the OSGi Developer Conference as a track within the EclipseCon conference. The OSGi specifications underlie the component model used within Eclipse and are also used in embedded devices and application servers. The Eclipse Equinox project is an OSGi framework that forms the core technology for Eclipse and Eclipse RCP. Attendees of the OSGi Developer Conference will have full access to all of the EclipseCon sessions and events.

[\[MORE INFO\]](#)

Intalio Donates BPMN Modeler to Eclipse Foundation



Intalio, an Open Source Business Process Management System (BPMS) company has donated its BPMN process modeler to the Eclipse Foundation. The technology is now available under the Eclipse Public License (EPL) and is part of the SOA Tools Platform (STP) project.

This contribution follows Intalio's donation of its EMF model comparator to the Eclipse Foundation earlier this year. The STP BPMN Modeler is one of three contributions made by Intalio to build the first Open Source BPMS. It complements the BPEL Engine donated to the Apache Software Foundation and the Tempo BPEL4People workflow framework hosted on Intalio.org. All three components form the foundation for Intalio|BPMS, the first BPM solution to support a Zero-Code development model.

"While some BPM vendors give their proprietary process modeling tool away, but charge for the necessary runtime components, we make our entire product available for free and give away the source code under Open Source licenses for its most critical components," said Ismael Ghalimi, founder and CEO of Intalio. "Furthermore, the building blocks for Intalio|BPMS provide the most faithful implementations of relevant industry standards for BPM, namely BPMN, BPEL and BPEL4People." "BPMN has become an important factor driving business-IT alignment," said Bruce Silver of BPMS Watch, "since it is intuitive enough to be used by business analysts, yet rich enough to generate powerful service-oriented implementations."

[\[MORE INFO\]](#)



JAX[®] INNOVATION AWARD 2007

20,000 Euros for your Innovation!



Product
website business idea
software technology
scientific contribution
blog closed source
open source

In the era of Web 2.0, it is time to seek out the best innovations that can take us to the next generation. The JAX Innovation Awards, endowed with a prize money of 20,000 Euros, has been constituted to throw the spotlight on the most striking contributions to the world of Java, Eclipse, and SOA. The awards will showcase technological innovations from the European market.

Who can participate?

Contributions can come from

- Individuals
- Companies
- Universities/Research institutes
- Other organizations

**Hand in your contributions
and win big!**

jax-award.com

Deadline: 19 March 2007



— New Releases —

Visual Editor for UltraLightClient and Eclipse 3.2 Available



Black Duck Software, a provider of software compliance management solutions, has announced the integration of protexIP/development 4.0 with IBM Rational Application Developer for WebSphere 7.0 and Rational Software Architect

7.0, part of the IBM Rational Software Delivery Platform 7.0, desktop products. This integration, the company touts, provides enterprise developers using the Eclipse development environment a fast new way to check code for compliance with hundreds of open source licenses as they work. Black Duck's validation of the integration between protexIP, which is touted to support the needs of many individuals and teams involved in software development, and Rational Application Developer offers WebSphere developers the ability to instantly check their code against the Black Duck KnowledgeBase, to ensure license compliance.

Black Duck has previously, achieved the IBM Ready for Rational software validation of protexIP integration for Rational Application Developer V6, and the Rational ClearCase software configuration management system. "Open source code permeates many organisations' software assets today, making compliance with the complex license terms governing the code a priority for corporate counsel and management. Now, with this updated integration, users of Rational Application Developer will be able to quickly assess if the code they want to use can be used and is in compliance with corporate policy—all without leaving the development environment and opening additional applications" said the company, in a statement. Black Duck Software's protexIP/development Enterprise Edition will be available for use with IBM Rational Application Developer 7.0 in early 2007. Developers will also be able to access protexIP directly through the open source Eclipse 3.2 development environment.

[\[MORE INFO\]](#)



Eclipse Plugins for Rapid Integration of Stream Processing Applications

StreamBase Systems, a provider of Complex Event Processing (CEP) software, has announced a new set of Eclipse-based plug-ins

that make it easy for developers to extend and rapidly integrate StreamBase applications into any enterprise environment. These plug-ins enable developers to connect new data feeds and custom adapters to StreamBase's stream processing engine in a fraction of the time typically required for extensive low-level custom-coding of real-time feed interfaces. "We've all seen mainstream adoption of stream processing for complex events intensify significantly over the last 12 months," said Dr. Michael Stonebraker, Founder and CTO of StreamBase. "This has driven the need for an open, easily accessible development environment based on accepted industry standards.

The company claims that the Eclipse-based plug-ins offer wizards and templates that dramatically cut the time required to integrate stream processing capabilities with existing messaging systems and real-time market data feeds. StreamBase developers can interface and connect StreamBase with existing infrastructures and systems including: XML message streams, financial market data feeds such as Reuters or Bloomberg, Exchanges such as NASDAQ and more. StreamBase's Eclipse-based environment for CEP will also provide the foundation for developers worldwide to take advantage of Eclipse capabilities including testing, debugging, and version control. Furthermore, the company claims its alignment with Eclipse's development environment enables StreamBase to be embedded or plugged into a variety of tools for application development, including those supported by OEM and ISV partners

[\[MORE INFO\]](#)



Intalio Releases Open Source Workflow Framework

Intalio, an Open Source BPMS company, has announced the release of the Tempo Workflow Framework under the Eclipse Public License.

"While BPEL is a very efficient process execution language, it does not provide any semantics for human workflow," said Ismael Ghalimi, founder and CEO of Intalio. "This need has been addressed by the BPEL4People paper jointly drafted by IBM and SAP, and today's release of the Tempo project marks the first Open Source implementation of this emerging standard."

Intalio's implementation of the BPEL4People model is made without any extensions or modifications to the standard BPEL 2.0 specification. Instead, ad-hoc task management services are

deployed on top of the J2EE platform and are made available as Web services through WSDL interfaces, while using standard BPEL processes for advanced workflow patterns such as multi-channel notifications and alert escalations, claimed the company

The release of Tempo under an Open Source license follows the donation of Intalio's BPMN modeler to the Eclipse Foundation and Intalio's BPEL engine to the Apache Software Foundation, both made earlier this year. All three components form the foundation for Intalio|BPMS, the first BPM solution to support a Zero-Code development model.

[\[MORE INFO\]](#)

Code Generator Eclipse Plug-in Launched



Bruno Braga, has announced the first international release of J2EE Spider, Version 0.1.1, an open source project about Java code generation. Spider is OS independent and for now, Braga is the only developer on the team.

The project explores advanced resources of code generation and aims to minimize the effort and time of J2EE projects, using the code generation.

The main concerns of the project are usability, integration with development platform, freedom to choose which code needs to be generated, incremental development generation and customisation of code templates to better fit the needs of the development team. Besides, other concern is support for several technologies like Struts, JSF and Tapestry. Only Struts is supported by the first version of J2EE Spider.

[\[MORE INFO\]](#)

AJDT 1.4.1 Released, Includes Support for AspectJ 1.5.3 on Eclipse 3.0-3.3



The AspectJ Development Tools (AJDT) team has announced the release of ADJT 1.4.1, providing numerous bug fixes, new features, along with includes support for Aspect 1.5.3. The AJDT project provides

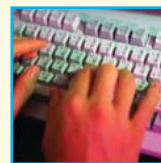
Eclipse platform based tool support for AOSD with AspectJ. Significant changes in the release include:

- Refactoring participant, that will search for and update references when a Java class is renamed
- Improved Binary Weaving support

- Improved AspectJ Build Properties
- Build Automation for AspectJ-enabled plug-ins
- Bug fixes

[\[MORE INFO\]](#)

Agitar Software Announces 'No Java Class Left Behind' Program



Agitar Software, a provider of developer testing solutions, has announced that its newly released product for Java unit testing – AgitarOne – is available free of charge to accredited educational institutions for teaching and academic research.

Agitar will also create and make available course material for introducing basic unit-testing principles in programming courses. The initiative will be carried out by Agitar Software Laboratories (AgitarLabs), Agitar's recently established division for research and advanced development. AgitarOne is a new integrated solution that aims to simplify Java unit testing by providing a complete set of tools in the market. It includes Agitar's software agitation technology for interactive exploratory testing, which enables automated generation of JUnit tests for thorough regression testing and enforces flexible and extensible code-rules that help eliminate error-prone coding patterns.

“Researchers and practitioners alike agree that having developers create and execute tests while they are writing the code is one of the most effective ways to prevent and find bugs. Today, however, most computer science students graduate with little or no exposure to the science and art of software testing,” said Alberto Savoia, Agitar's CTO and head of AgitarLabs. “We believe that the most effective way to get developers used to the theory and practice of testing is to teach them how to test at the same time they are learning how to program. The key goal of this initiative is to give both instructors and students easy access to superior teaching materials and technology, and to show them the huge benefits that developer testing can bring.” AgitarOne offers a server-based deployment model that model requires no configuration on the workstation and enables students to begin seeing meaningful results within minutes of downloading and installing an Eclipse-based client. Additionally, such a deployment model reduces the computing power requirements for students' workstations by offloading most of the computation to the distributed server, says the company.

[\[MORE INFO\]](#)

Introduction to the Generic Eclipse Modeling System

Developing a Graphical Modeling Tool for Eclipse

By Jules White, Douglas C. Schmidt, Andrey Nechypurenko, Egon Wuchner

Graphical Model-Driven Engineering (MDE) tools have become extremely popular in the development of applications for a large number of domains. In many cases, however, an organization does not have the resources or time available to develop a graphical modeling environment from scratch using the Eclipse Modeling Framework (EMF), Graphical Editor Framework (GEF), or Graphical Modeling Framework (GMF). In other situations, the complexity of the domain limits the feasibility of using a graphical model to describe a domain solution. The Generic Eclipse Modeling System (GEMS), which is part of the Eclipse Generative Modeling Technologies (GMT) project, helps developers rapidly create a graphical modeling tool from a visual language description or metamodel without any coding in third-generation languages. GEMS automatically generates the requisite EMF, GEF, and GMF code required to implement the editor from a metamodel. GEMS also provides extensive capabilities for expressing modeling guidance and performing optimization. Finally, graphical modeling tools created with GEMS automatically support complex capabilities, such as remote updating and querying, template creation, styling with Cascading Style Sheets (CSS), and model linking.



Introduction

Graphical Model-Driven Engineering (MDE) tools have become extremely popular in the development of applications for a large number of domains. A wide range of frameworks supports development of MDE tools for Eclipse. The **Eclipse Modeling Framework** (EMF) provides an object graph for representing models, as well as capabilities

for (de)serializing models in a number of formats, checking constraints, and generating various types of tree editors for use in Eclipse. The **Graphical Editor Framework** (GEF) and **Draw2D** provide the foundations for building graphical views for EMF and other models types. The **Graphical Model Framework** (GMF) for Eclipse provides a rich framework for reducing the complexity of developing an GEF/EMF-

Feature Introduction to the Generic Eclipse Modeling System

based modeling tool. These frameworks are designed to provide the degrees of flexibility needed to support products, such as IBM's Rational Architect.

MDE tools are rarely simple to develop, however, and are generally reserved only for the highest payoff domains. Regardless of the framework used, developers must still produce a significant amount of XML and Java code to construct a working graphical modeling tool. Smaller organizations tend to lack the required development expertise to implement a graphical modeling tool using EMF, GEF, and GMF. Even in situations where an organization can develop a graphical modeling tool from scratch, the time spent performing these activities subtracts from the time available for developing the true assets of these tools—the complex domain validation, code generation, optimization, and simulation capabilities. Moreover, there is a significant cost associated with maintaining a graphical modeling tool infrastructure, as understanding of the domain deepens and the tool and **domain-specific modeling language (DSML)** requirements change.

The **Generic Eclipse Modeling System (GEMS)**, a part of the Eclipse **Generative Modeling Technologies (GMT)** project, helps developers rapidly create a graphical modeling tool from a visual language description (metamodel) without any coding in third-generation languages. GEMS is an open source project, based on the Eclipse License, that has been developed in conjunction with Siemens CT SE2, IBM, and Prismtech. GEMS gives developers the ability to graphically describe the DSMLs they wish to create a modeling tool for and automatically generate the requisite EMF, GEF, and GMF code required to implement an Eclipse-based plug-in for creating and editing instances of the DSML. GEMS pulls together these various Eclipse modeling frameworks and uses standardized naming conventions, file formats, and other simplifications to make coding an editor from scratch unnecessary. Although GEMS uses multiple techniques to allow developers not to write code unless they want to, the decisions and default visualizations used by GEMS can still be overridden through the use of Cascading Style Sheets (CSS), extending GEMS extension-points and other facilities, such as model templates and remote updating mechanisms.

GEMS substantially reduces the cost of developing a graphical modeling tool by allowing developers to focus on the key aspects of their tool—the specification of the DSML and the intellectual assets built around the use of the

language. The infrastructure to create, edit, and constrain instances of the language is generated automatically by GEMS from the language specification. As the language specification changes, GEMS can regenerate the Java, EMF, GEF, GMF, and XML descriptors required to edit the new language. Moreover, GEMS allows the separation of language development, coding, as well as 'look and feel' development, such as changing how modeling elements appear based on domain analyses.

A GEMS-generated graphical modeling tool can have both static and dynamic visual behavior specified through stylesheets. Graphic designers or developers can create styles and icons that should be applied to elements when they are in specific states. For example, a developer building a tool for specifying the deployment of software components to nodes could develop separate icons and styles for drawing a component when it is deployed and undeployed. The CSS capabilities of GEMS will be discussed in detail in the Section 'Customizing the Look of the Modeling Tool with CSS'.

The process for developing and using a graphical modeling tool using GEMS is based on the Model Integrated Computing paradigm [5] developed at Vanderbilt University and originally implemented in the Generic Modeling Environment (GME) [6]. This process can be seen in **Figure 1**.

The first step for building a graphical modeling tool with GEMS is to define a metamodel for the DSML. This metamodel describes the graphical entities, connection types, attributes, and other visual syntax information needed by GEMS. In step two, the user invokes a code generator that produces the EMF, GEF, GMF, XML and so on, which are needed

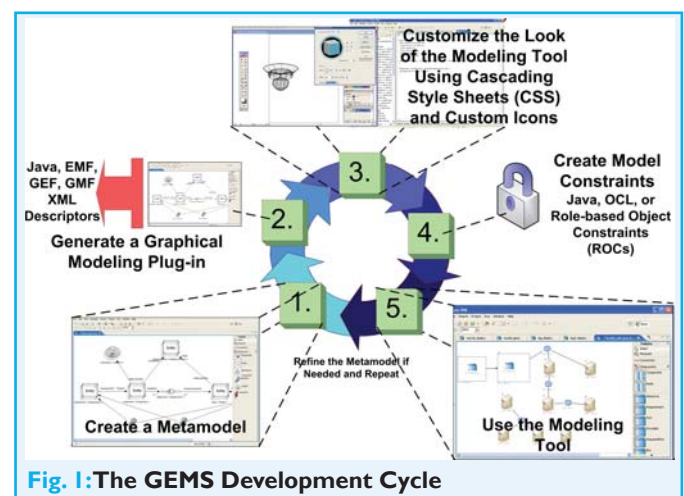


Fig. 1: The GEMS Development Cycle

to create a plug-in for editing the DSML described by the metamodel. Graphic designers or developers create custom icons and CSS styles for the DSML elements in the third step. For the fourth step, developers specify the constraints on the model that GEMS uses to ensure that only correct models are built with the model. The constraints generally involve domain information that requires a constraint language to express properly. Finally, domain experts use the modeling tool produced by GEMS to construct models of the domain. If the DSML is sufficiently expressive that no modifications are needed for the metamodel, model interpreters (or code generators) are developed to produce software artifacts (such as Java code or XML descriptors), run simulations, or execute the model. The following sections describe each of these steps in detail.

Creating a GEMS Metamodel

The core of a GEMS-based modeling tool is a metamodel describing the syntax of the DSML. As a case study, we describe the development of a tool, called AUTODeploy, for specifying the deployment of software components to nodes in a data center. AUTODeploy allows IT professionals to describe each software component they wish to deploy in their data center, the requirements of the node hosting each component, the nodes available in the data center, and the resources available on each node. AUTODeploy can also specify where components should be deployed by creating connections between components and nodes. Finally, this modeling tool can automatically deduce and create a valid deployment for all the components. **Figure 2**

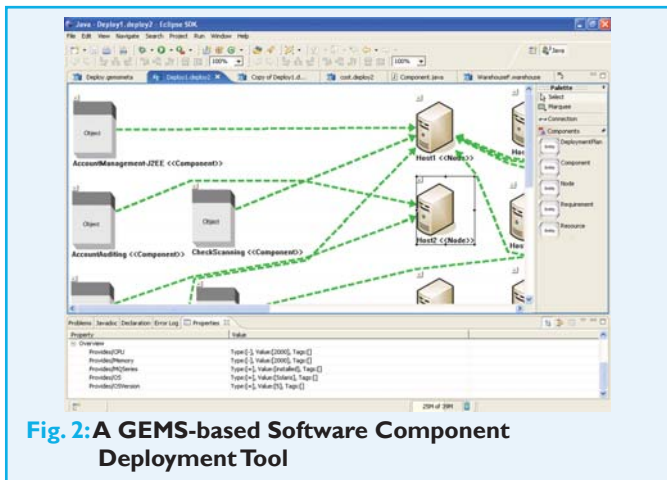


Fig. 2: A GEMS-based Software Component Deployment Tool

shows a screenshot for AUTODeploy that we will produce using GEMS.

Examining the finished AUTODeploy product we are creating helps clarify how various parts of the GEMS development cycle fit together and map to AUTODeploy, which has five different types of model entities or elements. The first entity is the *DeploymentPlan*, which is represented as the main white canvas containing the *Nodes* and *Components*. The *DeploymentPlan* is the root entity in our model. Node and Component entities can be seen on the right and left hand sides of the model, respectively. Two more entity types are present but not visible in the screenshot. *Requirements* are child entities of Components that can be seen by expanding a Component by clicking on the button in the upper left hand corner of each Component. *Resources* are child entities of Nodes and are also not visible. We can, however, see evidence of the Resources in the properties pane of the tool.

The *Overview* attribute in the properties pane visible in the bottom of **Figure 2** provides a brief outline of the children contained by each entity and the attributes of the children. CPU, GeoDB, and OS are all Resources contained by Host3. The finished AUTODeploy tool contains connections between Components and Nodes. These are Deployment connections that were defined in the metamodel. **Figure 3** shows the complete metamodel for our software component deployment tool.

The *Entity* elements in the metamodel correspond to the entities we described in AUTODeploy. The metamodel can

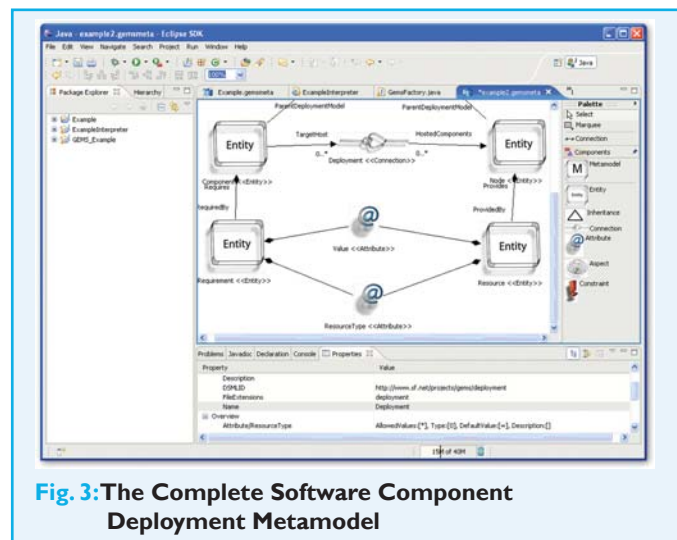


Fig. 3: The Complete Software Component Deployment Metamodel

Feature Introduction to the Generic Eclipse Modeling System

be viewed as a class diagram for the instances of the model entities seen in AUTO Deploy. The Entity elements in Figure 3 correspond to the types in our DSML. The *Attributes*, visualized with the '@' sign, represent the properties of the types that can be set in the property pane. The *Connection* element, called Deployment, corresponds to the Deployment connection type in AUTODeploy. **Figures 4, 5, 6, and 7** illustrate these mappings.

Entities can have *Inheritance* relationships between them, although this example does not show this feature. Inheritance relationships are created by adding an Inheritance element to the

model and connecting it to the parent type and derived type(s). Any attributes, connections, or containment relationships specified by the parent are inherited by derived types. *Aspects* are another metamodel element that can be defined to group sets of Entities and Connections into separate views on the model. Entity types are associated with an Aspect element in the metamodel through containment relationships. In the generated modeling tool, Aspects appear as unique views that can be used to filter the currently visible Entity and Connection types. Custom CSS styles can be defined for an element so that its visual appearance changes when the aspect changes.

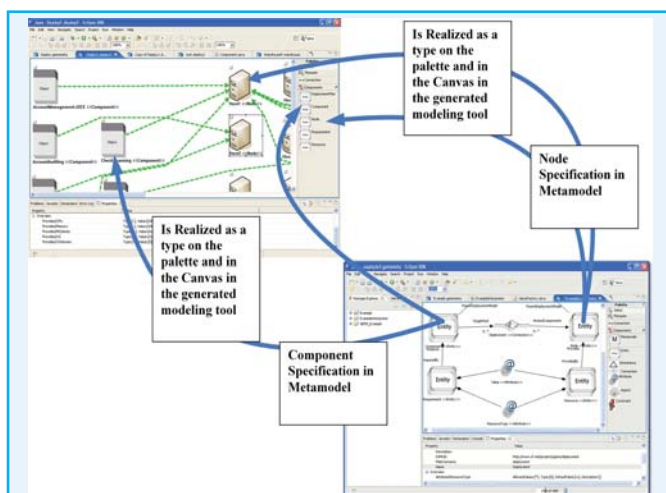


Fig. 4: Realization of Entities in AUTODeploy

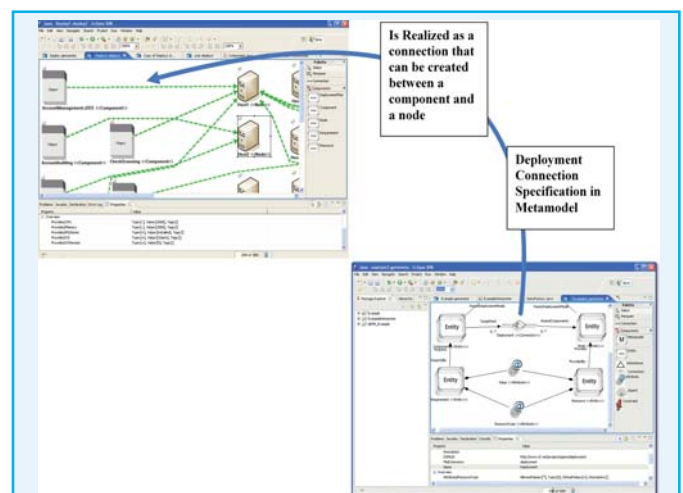


Fig. 5: Realization of Connection Entities

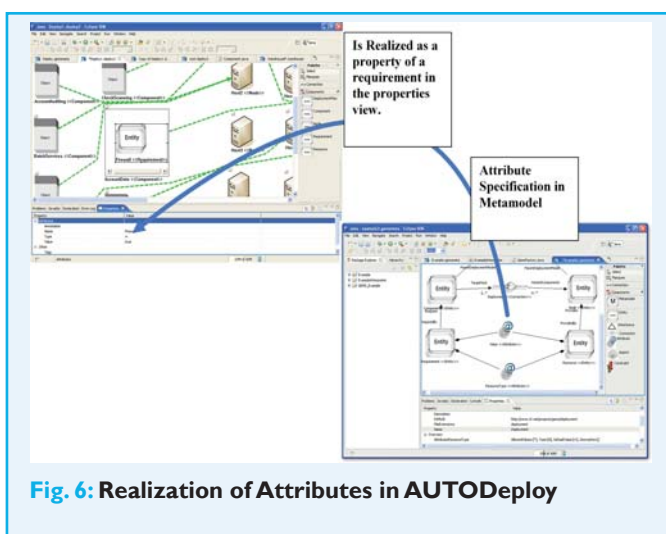


Fig. 6: Realization of Attributes in AUTODeploy

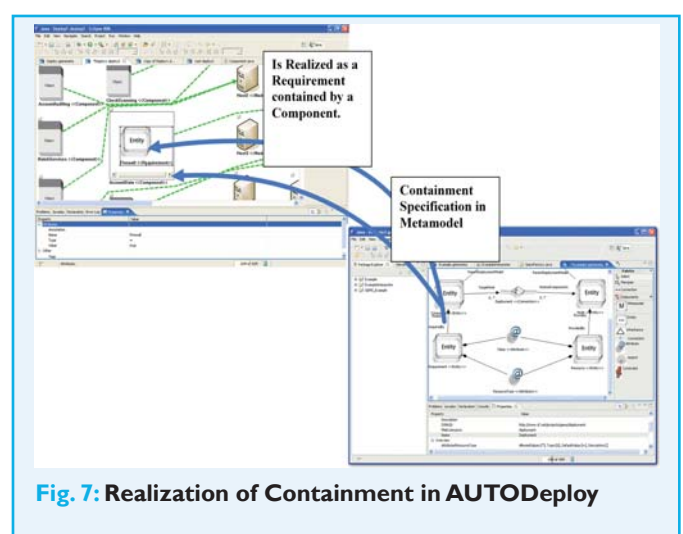


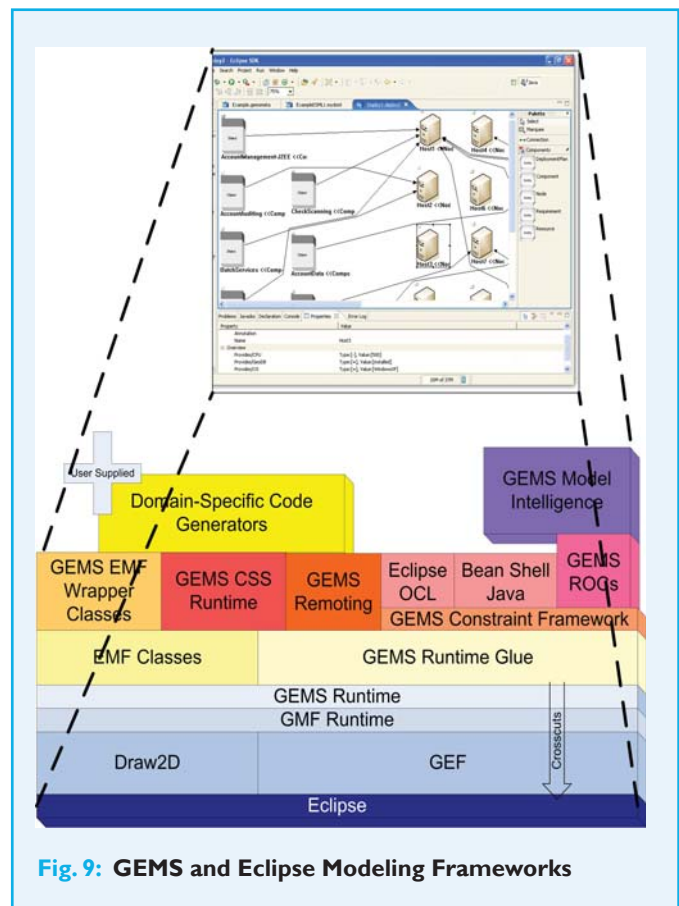
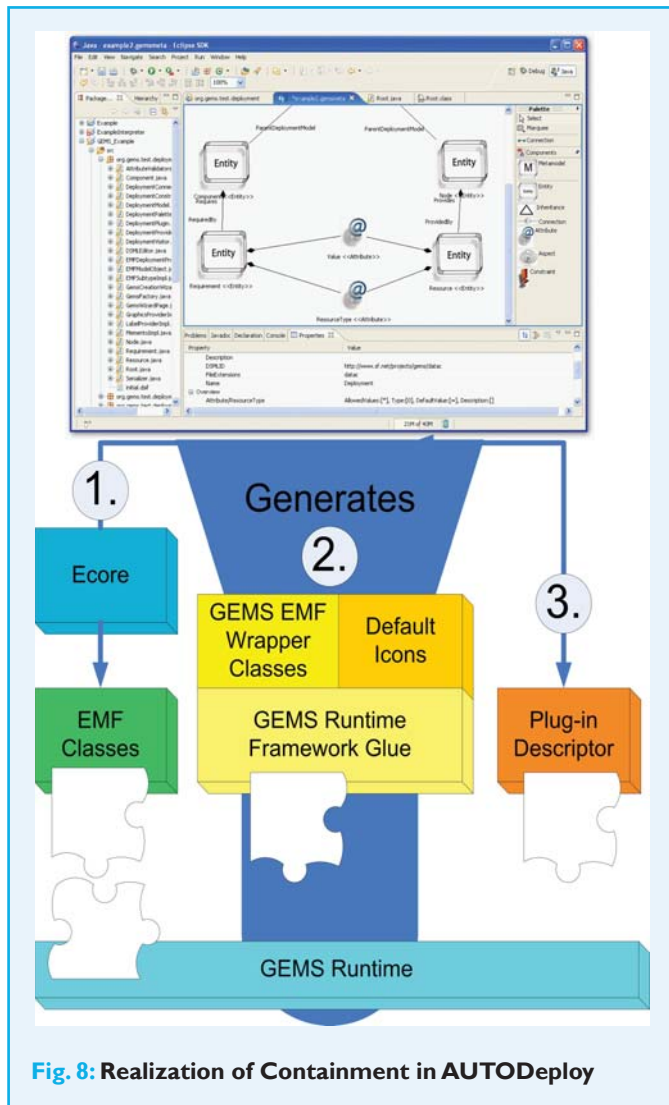
Fig. 7: Realization of Containment in AUTODeploy

Generating a Graphical Modeling Tool Using GEMS

After creating a metamodel, GEMS's DSML plug-in generator can be invoked to create the modeling tool. The code generator first traverses the various entities in the model and generates an Ecore model, which is a set of EMF objects that represent the metamodel or syntax of the visual language, to implement the metamodel. GEMS then invokes the appropriate EMF code generators to produce EMF classes that implement the Ecore definition. This process can be seen in Step 1 of [Figure 8](#).

The generated EMF classes are used as the object graph underlying AUTODeploy. These classes automate key aspects of serialization and de-serialization. By default, GEMS leverages EMF's ability to save EMF models to XMI. Other serializers can be plugged into persist models in a database or use an alternate file format. The EMF object graph also allows GEMS to leverage the libraries available for Eclipse to check Object Constraint Language (OCL) constraints against a model.

The second set of code generated by GEMS are the classes required to plug the generated EMF code into the GEMS runtime framework. The GEMS runtime provides a layer built on top of GEF (and soon GMF) that provides higher level capabilities, such as applying CSS styles to elements, exposing remote update mechanisms, and providing constraint solver modeling intelligence. The relationship between the generated artifacts, GEMS runtime, and Eclipse modeling frameworks can be seen in [Figure 9](#).



The GEMS runtime provides numerous extension points for adding custom functionality to the generated modeling tool. Extension points are available for adding actions that can be triggered by OCL, Java or role-based object constraint assertions on the model, customization of menus, custom code generators, remoting mechanisms, custom serializers, and many other features. As the underlying Eclipse modeling technologies evolve, GEMS continues to incorporate and expose their new features.

The third set of code artifacts, generated by GEMS, are the various XML descriptors, build specifications, classpath directives, and icons required to integrate the generated tool into Eclipse as a plug-in. When a modeling tool is generated into a Java project, these various artifacts configure the project properly to build the plug-in and make it visible for testing with the runtime workbench. The build artifacts also configure the project so that it can be exported properly as an Eclipse modeling tool plug-in.

Customizing the Look of the Modeling Tool Using GEMS

One goal of GEMS is to avoid requiring developers to use third-generation languages to write any graphics code required to implement a modeling tool. In most cases, however, simply providing a stock set of visualizations will not suffice. GEMS allows developers to customize the look and feel of their modeling tools. It supports this capability by allowing developers to change the default icons visible on the palette and in the model, which enables developers to quickly create impressive visualizations that match the domain notations.

Swapping icons isn't the only mechanism GEMS provides to customize the look and feel of modeling tools like AUTODeploy. Developers can use CSS style sheets to change fonts, colors, backgrounds, background images, line styles, and many more features of a GEMS-based modeling tool. If an organization has a graphic design department, the customization of the look is similar enough to using HTML and CSS that they can often handle this effort. **Figure 10** shows the application of a CSS line style to change the line to a four-pixel, dotted, green line.

CSS styles are applied to elements using traditional CSS selectors. The key difference is that selectors refer to the roles and types specified in the metamodel. For example, to make all the Components that are deployed have a different

style than un-deployed Nodes, a style could be created with a selector that matches the source role of a Deployment connection. After adding this style, any Component that serves as the source of a Deployment connection will have the style applied.

Even more complex visual behaviors can be created by leveraging a feature called TAGS in GEMS. TAGS are textual markup that can be added to modeling elements to denote that they have a certain property. For example, AUTODeploy could provide visual queues to modelers by changing the background color of nodes that cannot host any more components to red. Two pieces of code are required to create this complex behavior based on domain analysis:

- An OCL, Java, or role-based object constraint trigger must be added to include a 'NodeFull' tag to nodes whose resources cannot support any more Components
- Create a CSS style that matches Nodes with the 'NodeFull' tag

The GEMS TAGS facility supports the combination of domain analysis with CSS styles to rapidly develop complex domain-specific visual behaviors, such as changing the icon for a node that doesn't have sufficient resources to host anymore components or highlighting green nodes that have more than a predefined level of slack on their CPU.

Adding Domain Constraints

A key benefit of MDE tools is their ability to capture and enforce domain constraints. The GEMS constraint framework supports a number of constraint types,

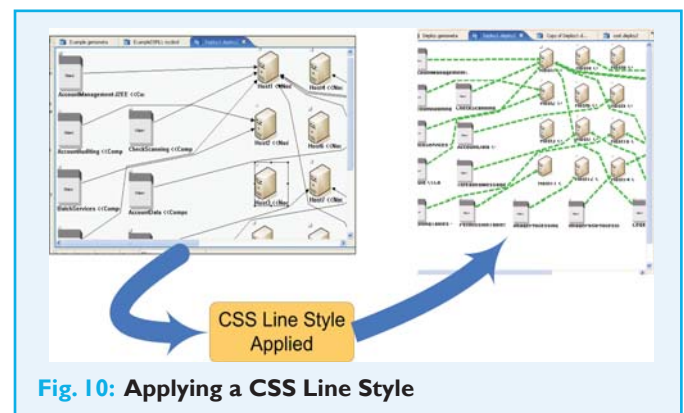


Fig. 10: Applying a CSS Line Style

including OCL, Java, and role-based object constraints. Other constraint languages can be plugged into GEMS by extending various extension points in the plug-in.

GEMS views constraints more as triggers than actual constraints since this allows constraints to perform a number of actions to guarantee model correctness, such as showing a warning message, vetoing a model change, or adding elements to fix the error. In GEMS, a domain analysis is performed using one of the constraint languages, such as OCL, and when the analysis indicates that a constraint has been violated, an action is triggered. For a traditional constraint, the action can veto the modeling event that produced the constraint violation and roll back any changes that it caused. A trigger can also result in a message popping up above the modeling element that has violated the constraint. Much more complex actions can take place as well. For example, a constraint violation can trigger an action that runs automated diagnostics to identify why the event was generated, as seen in [Figure 11](#).

Model Intelligence

A unique feature of GEMS that sets it apart from other MDE tools is its integration of Role-based Object Constraints (ROCs). ROCs allow developers to describe declarative domain constraints, just like they would with OCL, except that GEMS can leverage its built-in constraint solvers to find solutions for many types of constraint problems. To put it another way, if you tell the GEMS-based modeling tool about what the rules are for a correct solution, it can find the solution for you.

For example, in AUTODeploy, each Component must be deployed to a node that supports the proper configuration of middleware, OS, OS version, databases, and so on. These constraints are modeled as Requirements that must be matched with $>$, $<$, or $=$ against a value on a Resource contained by a Node. For example, you could have the requirements, $OSVersion > 2.3$, $MultiCoreCPU = true$, $FireWallInstalled = true$, or $HeatProduction < 25$. In AUTODeploy, users can add arbitrary name, comparator, value requirements that must be matched against the Resources contained by a Node.

A realistic deployment model for a data center will likely contain hundreds of servers and hundreds or thousands of software components. Human modelers clearly can't

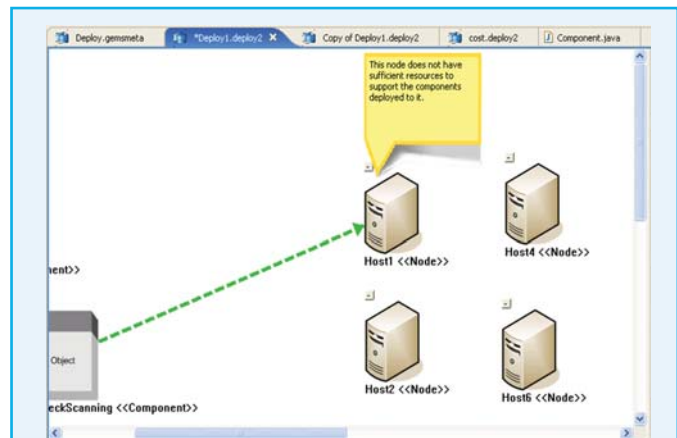


Fig. 11: A Constraint Violation Triggering a Popup Warning

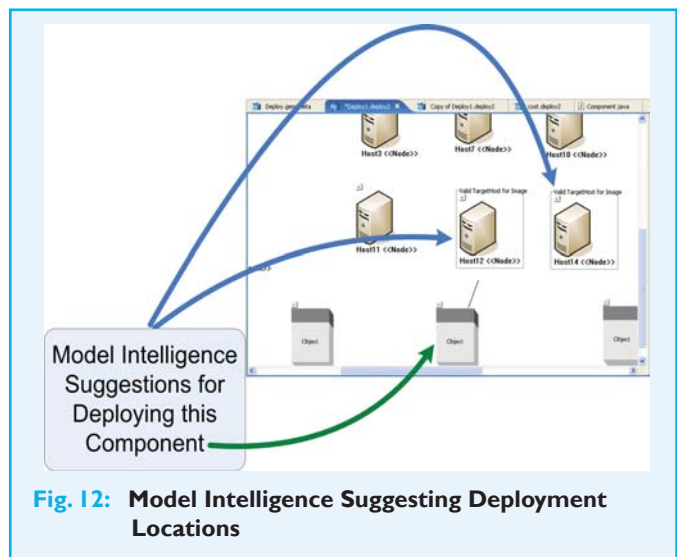


Fig. 12: Model Intelligence Suggesting Deployment Locations

point and click their way through a deployment of this size. The GEMS Model Intelligence and ROCs are designed to handle problems at this scale.

Model Intelligence provides a modeling tool like AUTODeploy with two key capabilities. First, it allows the tool to show a modeler the valid ways of satisfying a constraint. For example, in AUTODeploy, when a user places the mouse over a Component, Model Intelligence can highlight the Nodes that satisfy the configuration requirements for the Component. This type of suggestion can be seen in [Figure 12](#).

Feature Introduction to the Generic Eclipse Modeling System

Suggesting local modeling decisions is certainly a huge improvement over merely constraint checking a manually produced solution. The real benefit of Model Intelligence becomes evident when it is combined with batch processing. As described previously, large-scale models are too complex to manage manually, particularly if the model has complex global constraints, such as resource constraints. GEMS provides a facility for combining batch processing with Model Intelligence to perform complex assignments automatically, such as finding a target host for every Component in a model.

To see how this could work using GEMS, we will add the following ROC rules written in Prolog to AUTODeploy:

```
compare_value(V1,V2,'>') :- V1 > V2.  
compare_value(V1,V2,'<') :- V1 < V2.  
compare_value(V1,V1,'=')
```

```
matches_resource(Req,Resources) :-  
  member(Res,Resources),  
  self_name(Req,RName),  
  self_name(Res,RName),  
  self_type(Req,Type),  
  self_value(Req,Rqv),  
  self_value(Res,Rsv),  
  comparevalue(Rsv,Rqv,Type).
```

```
can_deploy_to(Componentid,Nodeid) :-  
  self_type(Componentid,component),  
  self_type(Nodeid,node),  
  self_requires(Componentid,Requirements),  
  self_provides(Nodeid,Resources),  
  forall(member(Req,Requirements),matches_resource(Req,Resources)).
```

These rules implement exactly the requirement to resource matching that we described earlier. They tell Model Intelligence to only deploy a Component to a Node if all of the Requirements are met by the Node. We can then make the rule 'can_deploy_to' invocable as a batch process. The result of running this Model Intelligence batch processor can be seen in [Figure 13](#).

The ROCs that Model Intelligence uses are a programming paradigm built on top of constraint solvers. The default solver packaged with GEMS uses Prolog since it provides a large number of constraint solver and optimizer implementations,

Douglas C. Schmidt



is a Full Professor in the Electrical Engineering and Computer Science (EECS) Department, Associate Chair of the Computer Science and Engineering program, and a Senior Research Scientist at the Institute for Software Integrated Systems (ISIS) at Vanderbilt University, Nashville, TN. For the past two decades, he has led pioneering research on patterns, optimization techniques, and empirical analyses of object-oriented and component-based frameworks and model-driven development tools that facilitate the development of distributed middleware and applications.

Andrey Nechypurenko



works at Siemens Corporate Technology SE2 in Munich, Germany.

Egon Wuchner



works Siemens Corporate Technology SE2 in Munich, Germany.

Jules White



is a researcher in the Distributed Object Computing (DOC) group at Vanderbilt University's Institute for Software Integrated Systems (ISIS). He is the head of development for the Generic Eclipse Modeling System (GEMS).

Introduction to the Generic Eclipse Modeling System Feature

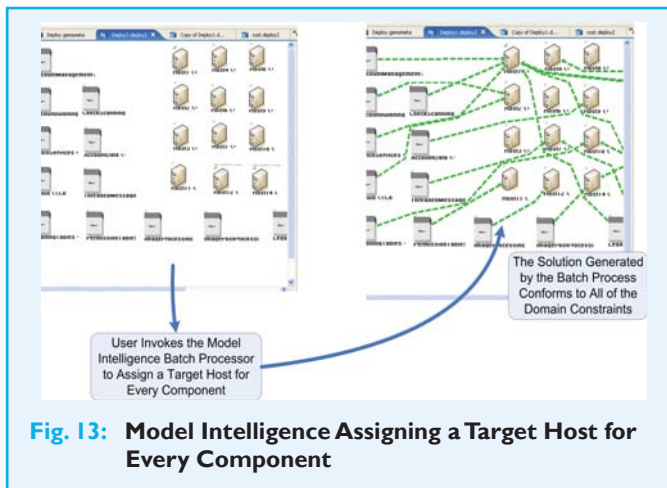


Fig. 13: Model Intelligence Assigning a Target Host for Every Component

Resources & References

- [1] GEMS: <http://www.eclipse.org/gmt/gems>
- [2] GEMS Download and Development: <http://www.sf.net/projects/gems>
- [3] GEMS Research at the Distributed Object Computing Group <http://www.dre.vanderbilt.edu/~jules/gems.htm>
- [4] The Distributed Object Computing Group <http://www.dre.vanderbilt.edu>
- [5] Model-Integrated Computing at ISIS: <http://www.isis.vanderbilt.edu/mic.html>
- [6] Generic Modeling Environment (GME): <http://www.isis.vanderbilt.edu/Projects/gme>

an easy to use declarative language, and good performance. ROCs allow users to describe constraints using domain-specific notation, while at the same time allowing existing complex Prolog solvers and algorithms to be plugged in. ROCs is an extensible paradigm not limited solely to Prolog-based solvers. New solvers, such as highly optimized bin-packers, simplex methods, or other libraries can be incorporated into the paradigm. Plugging a solver into ROCs is a non-trivial task, however, and is rarely needed. In the software component deployment modeling tool that we developed with GEMS, complex Prolog-based solvers have been developed for resource constraints. We are in the process of templating and incorporating these complex solvers into the base ROCs installation.

Concluding Remarks

This article has described the powerful features available for building graphical modeling tools with GEMS. The article has just scratched the surface of what GEMS provides and many features, such as remote updating and model templating have not been addressed, due to space limitations. For more information on GEMS, please refer the project web site ^[1].

www.jaxmag.com



Come, See What's Brewing

The Premier Online Resource for Java, Apache,
XML and Web Services

- Fresh Brew
- Jax Hojo
- Coffea Works
- Book Club

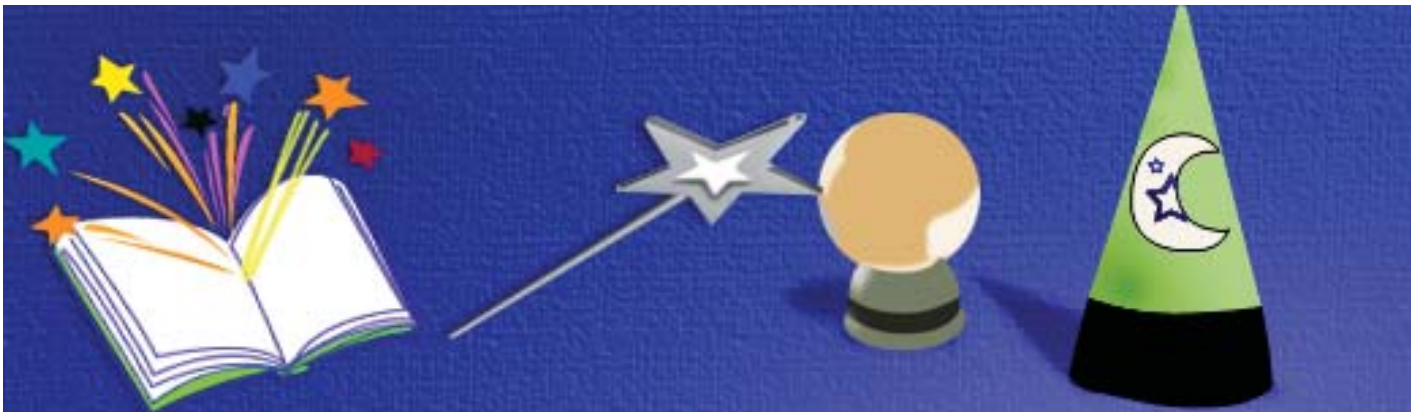
 **jax** magazine

Dynamic Wizard Modeling with GMF

Using GMF to Build a Dynamic Wizard Framework and a Graphical Editor

By Rajkumar C Madhuram

Developing a graphical editor is generally very complicated and requires lot of effort. There are few frameworks available for writing graphical editors in Java. The prominent open source frameworks are JHotDraw (which is Swing based) and GEF (which is SWT/Jface-based). While they provide sophisticated tools for graphical development, the painstaking work of modeling the domain and mapping to graphical elements is left to the user. Graphical Modeling Framework (GMF) bridges this gap nicely. In the article, I will take you through an end-to-end demonstration of GMF. To achieve that, first we will create a framework for meta-data driven JFace wizards. Next, we will see how to use GMF to build a graphical editor for this framework.



Introduction

Have you ever used a graphical editor and wondered what it takes to create one? In this article, we will see how to develop a graphical editor from scratch, using the Graphical Modeling Framework (GMF).

Before GMF, there was the Graphical Editing Framework (GEF) [1][2]. GEF offers a powerful and flexible platform for developing graphical editors. It makes use of the Model-View-Controller (MVC) pattern heavily. Each item on the canvas has three distinctive features, a) **model**, that is the data pertaining to the item, b) **view**, that is the visual representation and c) **controller**, that ties the model and view together.

One limitation with GEF is that it treats models as plain

Java objects (POJOs). Hence the programmer needs to fill in the semantics associated with the model and map it to the relevant edit parts. This is a fairly cumbersome process, as there may be hundreds of types of objects on the canvas. Another problem the programmer has to deal with is the serialization and de-serialization of data. The model has to be stored in a file, typically in XML format. Along with the model, the graphical information also has to be persisted. For instance, if an object is represented as a rectangle, we need to store its placement, width, height, color, and so on. Mapping this information to the model information is also a tedious task.

Eclipse Modeling Framework (EMF) is a powerful framework for modeling data. It provides facilities for code

generation, meta-data query/manipulations, editors, and serialization. As I pointed out earlier, these are features that would complement GEF in building a graphical editor. The Graphical Modeling Framework (GMF) is based on GEF and EMF, and it provides a comprehensive solution for building graphical editors.

The graphical editor we are going to develop will model the page flow of JFace wizard pages. **Figure 1** shows the outline of the steps involved. First we will start by defining the data model (**ecore**). From the data model, we will proceed to create the model classes. Using the generated classes, we will build the dynamic wizard framework. After that, we will create the GMF graphical and tooling model, which would enable us to create the graphical editor.

The graphical editor that is created will be an Eclipse plug-in. The user will be able to install the plug-in in her instance, create any project (Java or otherwise), create the wizard diagrams, and run the wizards by right clicking on the wizard file and selecting Run Wizard menu option.

Plug-in	Description
wizard	The main plug-in where the models reside. It contains the EMF generated model code in src/ directory. Our custom figures for GMF reside under custom/ directory.
wizard.diagram	Auto generated plug-in that contains GEF code for graphical editor.
wizard.edit	Auto generated plug-in that contains providers for EMF models.
wizard.	Our plug-in that contains code for the JFace Wizard framework.
wizard.runner	Our plug-in that contributes the Run Wizard menu option and also runs the selected wizard diagram.

Table 1: List of Plug-ins Involved in the Graphical Editor

It is recommended to have a read through the GMF tutorial^[4] (all the parts) before starting on this. Besides, you'll need to install GMF on Eclipse 3.2.1.

Dynamic Wizard Framework

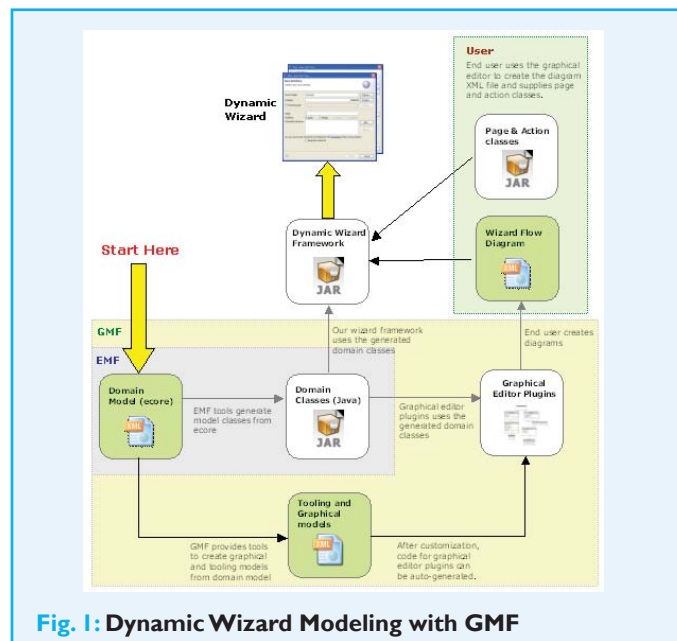
JFace has a powerful framework for creating wizards^[5]. However, the pages have to be hardwired to the wizard, using the `addPage()` method call. If you are into web

The article demonstrates the use of GMF in building a graphical editor. First a framework for JFace wizards is developed, that allows one to dynamically control the page flow. Next the author maps the domain to GMF elements and shows how to construct a graphical editor with very little effort.

development, this is not very different from how page transitions had to be hard-coded in JSP pages. One problem with this approach is that the re-usability of the pages is severely limited. To promote reuse, we could model the page description and the transitions using XML (similar to the Struts framework for Web).

Our dynamic wizard framework is essentially a finite state machine (FSM) ^[6]. Each page that is displayed is a state. We have two types of states—**display states** and **action states**. Display states have user interface associated with them, whereas action states do not. States throw events, and based on the event thrown, they transition to the next state.

When the user clicks the Next> button of the wizard, the engine is queried for the next page. It looks at the state diagram and determines the next page and returns it. Action states can be present in between display states. The



Quick start

The code for the examples detailed in this article is included in the source code pack downloaded along with the magazine PDF. Follow the instructions specified in the file setup.doc to install and run the samples. You can also visit <http://www.impigertech.com/wizard/>.

engine executes the actions as it encounters them in the course of transition between display states. See [Figure 1](#).

The Domain Model

To specify the model, first create a new GMF project. The supplied plug-in project 'wizard' is a GMF project. To create this, select New→Other→Graphical Modeling Framework→New GMF Project. All this does is to create a directory inside the project called model and add the required plug-in dependencies. We have all the model artifacts inside this directory.

Inside the model directory, you will see the file wizard.ecore. The.ecore file contains data about the model, and hence is called as the meta-model. It uses the UML notation. To create this file from scratch, right click on the model directory and select New→Other→Example EMF Model Creation Wizards →Ecore Model. The.ecore model can be developed using the Ecore Diagram Editor. To use this, however, you need to install the GMF Examples feature.

Inside the model directory, there is also a file called wizard.ecore_diagram. When this file is opened with the diagram editor, the model looks like that shown in [Figure 2](#). It should not be confused with the dynamic wizard framework class model. This model specifies the domain and will dictate what will be contained in the XML file that is produced when serializing the diagram.

We will now examine the classes that are modeled. The class Wizard is the container of everything. It contains zero or more states and transitions. There are two types of states, a DisplayState which has a visual page associated with it, and an ActionState that just performs an action and transition to another display or action state. There is also a special marker state called StartState, which signifies the start of the wizard. It must have a transition to a display state (an implicit assumption that is not modeled for the sake of simplicity).

Each state has a 'name' attribute that serves as an identifier for the state, and hence is unique. It also contains a 'class' attribute. The engine loads the states (pages and actions) at runtime based

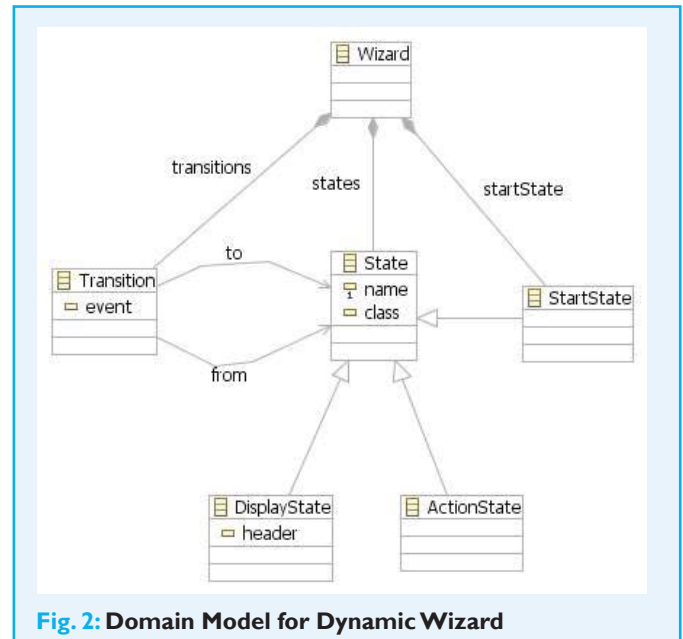


Fig. 2: Domain Model for Dynamic Wizard

on the class attribute. Thus, the page and action classes should implement interfaces defined in the framework. You can find more details about the framework implementation in the following sections. The DisplayState class has an attribute 'header', the value of which is displayed as the title of the wizard page.

From this meta-model, we can create the model classes. This is a two-step process: 1) Create genmodel and 2) Generate code. The first step is to create another XML file called the Generator Model file (**genmodel**). The objective of this file is to provide parameters to customize code generation. Fortunately, EMF provides a wizard to create this easily. Select File→New→Other→Eclipse Modeling Framework→EMF Model. Select the model/ directory and specify 'wizard.genmodel' for the filename. In the 'Model Importers' page, select 'Ecore' model. In the next page, select the wizard.ecore file from the workspace. Click on Load to import the model. Now click Next and Finish.

The genmodel file will be opened in a tree editor. The generator can generate multiple code for model, a tree editor for building the XML file based on your model and test code as well, all from a single domain model (.ecore). There are plenty of customizations possible to completely control the generated code. You can select a node in the genmodel and look at the properties view to get the idea. Now, right click on the root node Wizard and select Generate Model Code. This will

create all the code for manipulating the model elements and a factory class to create them in the `src/` directory of the wizard plug-in project.

Wizard Framework Implementation

In this section, we will look at the dynamic wizard framework implementation. Feel free to skim over this if you are not too concerned about the implementation aspects. The code for the framework is contained in `wizard.framework` plug-in. **Figure 3** shows the implementation of `DynaWizard`, which extends the `JFace Wizard` class. The constructor accepts a `Wizard` object, that is the root of the domain model we constructed in the last section. If you look into the code of `DynaWizard`, method `addPages()`, you will notice that we pre-load all the display and action state classes.

The `DynaWizard` class also contains three Hashmaps. One maps the display states to wizard pages (`org.eclipse.jface.wizard.IWizardPage`). Another one maps the action states to `DynaAction` objects. The last one holds session variables. The concept of session here is not different from the session in web applications. It contains information that needs to be shared across pages and actions.

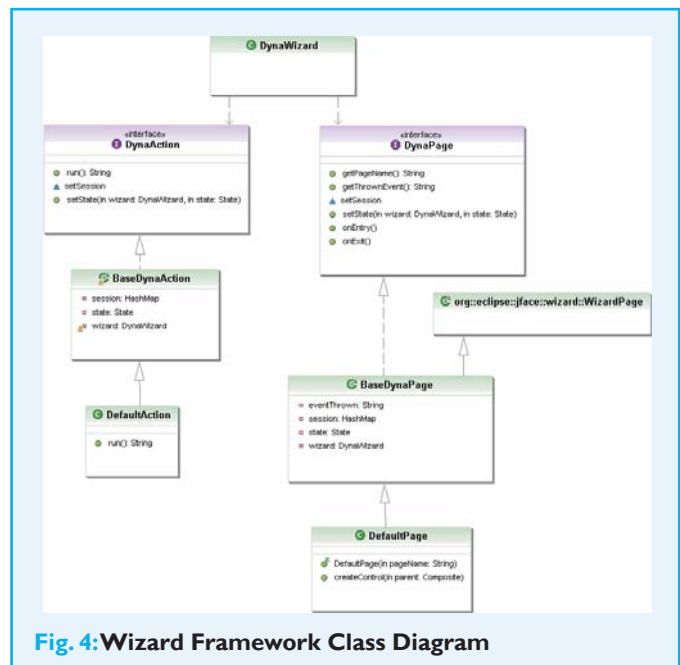
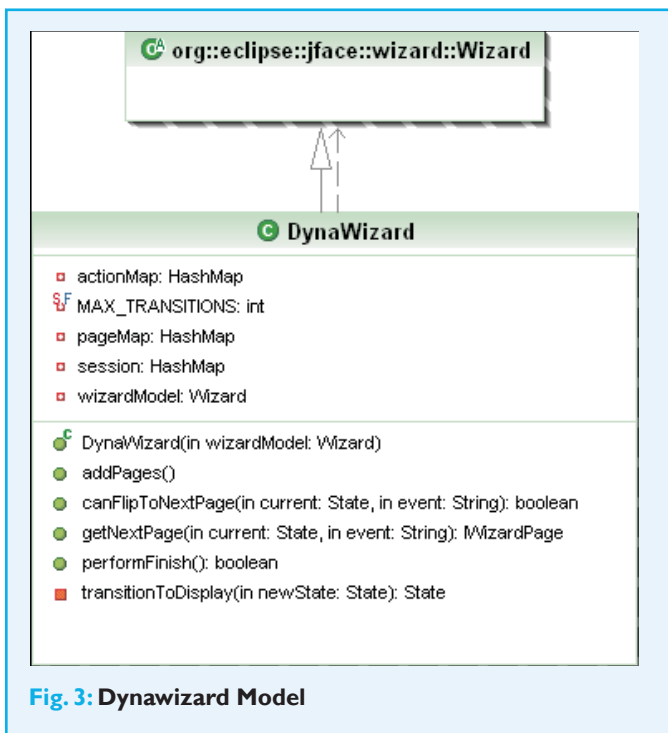
Now let's take a look at the method that makes the framework

dynamic, `getNextPage(State current, String event)`. It is called from `BaseDynaPage` where it overrides `getNextPage()` method. It looks at transitions from the current state and executes any actions until it reaches a display state. When it finds the next display state, it returns the corresponding page.

Figure 4 shows the class diagram for the dynamic wizard framework. The interfaces `DynaPage` and `DynaAction` has two important methods, `setSession()` which provides the session to the underlying page or action, and `setState()` which provides the domain model and the wizard.

The `BaseDynaPage` class is a binding to `JFace` wizard page. It queries the wizard for `canFlipToNextPage()` and `getNextPage()` calls. The method `throwEvent()` can be called by the widgets in the subclasses to throw events. It registers the event and calls `updateButtons()` method of the wizard container. This in turns calls the `canFlipToNextPage()` method and determines if the Next> button can be enabled. The `DefaultPage` is a page that displays the state name. Just to spice it up, it also contains a browser with URL set to the wikipedia entry corresponding to the state name!. It throws an event 'next' by default.

Action classes may extend `BaseDynaAction` and implement the `run()` method. Actions throw events by returning the event name from the `run()` method. The `DefaultAction` is a simple implementation that throws 'ok' event always.



Feature Dynamic Wizard Modeling with GMF

Wizard Runner

The plug-in `wizard.runner` takes care of running the dynamic wizard. It contributes a menu item 'Run Wizard' when you *right click* on the wizard. After creating the wizard.ecore model from the file, the class `wizard.runner.popup.actions.RunnerActionDelegate` creates a new URL classloader by adding the output directory of the project in which the wizard is present. This enables the wizard framework to load the page and action classes dynamically, as specified by the user.

GMF Models

The graphical editor produced by GMF is essentially a graph (as in graph theory) modeller. It has two main types of objects—nodes and links. A node is typically associated with a domain object and a link connects two domain objects. The editor is produced by defining a series of models. [Figure 5](#) shows the various models involved.

One of the strengths of GMF is the separation of concerns. The domain model is kept clean and separate, as we saw in the previous section. The graphical definition describes all the graphical elements in the editor. The shapes of nodes and connections and their properties are all described here. No association is yet made with the domain model. Then there is the tooling model. It describes the various tool elements like the palette, actions, and so on.

The mapping model brings together the domain model, graphical, and tooling definitions. It primarily consists of two main types of mapping—**node** and **link mappings**. A node mapping maps the domain element to graphical definition (for example, rectangle, ellipse or custom figure). It also maps the tool used to create this object on the canvas (typically a palette entry). A link mapping is very similar, but it maps the domain element

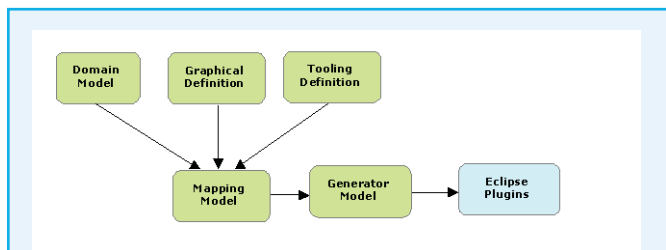


Fig. 5: GMF Models

to the connection figure (the polyline connection, for example) graphically.

The generator model consists of customization to code generation, like the edit part class name and plug-in name. Using the generator model, GMF creates all the required plug-ins for the graphical editor.

Graphical Definition Model (gmfgraph)

The GMFGraph file defines all the graphical elements present in the editor. I'll describe the process by which this file can be created. Select `File→New→Other→Graphical Modeling Framework→GMFGraph Simple Model`. Set the parent folder to 'wizard/model', the directory where the.ecore file is placed. Select a `gmfgraph` file name (don't overwrite `wizard.gmfgraph`). Select `/wizard/model/wizard.ecore` for the domain model file name. Clicking `Next >` will bring up a page to select the model elements to process, as shown in [Figure 6](#).

Select 'Wizard' for the diagram element. Recall that Wizard is the top-level container node in the model definition. It is mapped to the diagram canvas, which contains all the objects. Uncheck 'State' and all its attributes, as we are not interested in having a visual representation for the base class.

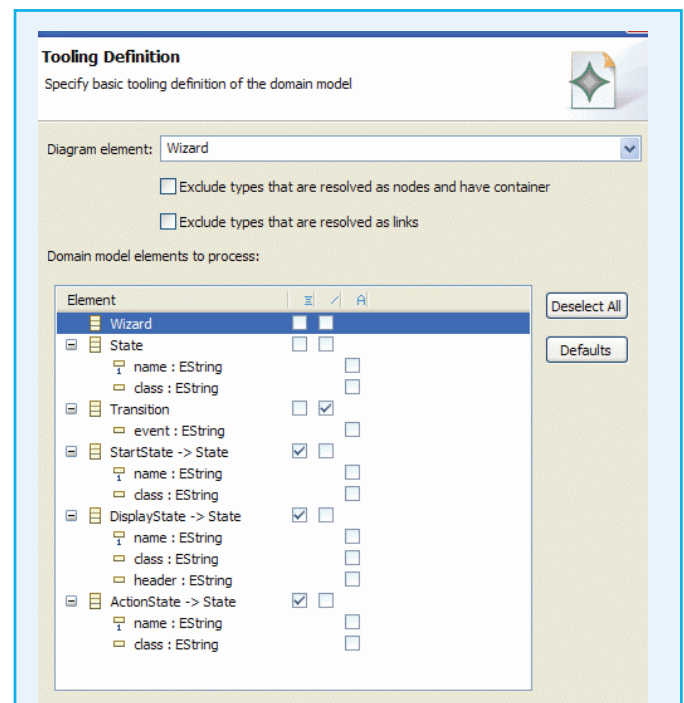


Fig. 6: GMF Graphical Definition Creation

Notice how the 'Transition' object is automatically resolved as a link. Also uncheck the attributes for 'StartState' as we are not interested in displaying them (it is just a marker state).

Let's take a look at the `model/wizard.gmfgraph` file in the 'wizard' project. It contains the graphical definitions for our editor. It is different from the one generated by the process described earlier, since I've customized it to suit the presentation needs. Look carefully at the tree structure. Below the 'Canvas wizard' node is the 'Figure Gallery Default'. This node contains all the figure definitions. There are four figures defined. Below the figure gallery node, there is a set of nodes, diagram labels and connections corresponding to the diagram model elements. Keep in mind that these are not the same as the domain model. We need to map these diagram model elements to the figures. You can click on any of these elements and observe the 'Figure' property in the Properties view. For instance, you can see that the node 'Node StartState' is mapped to the figure 'Ellipse StartStateFigure'.

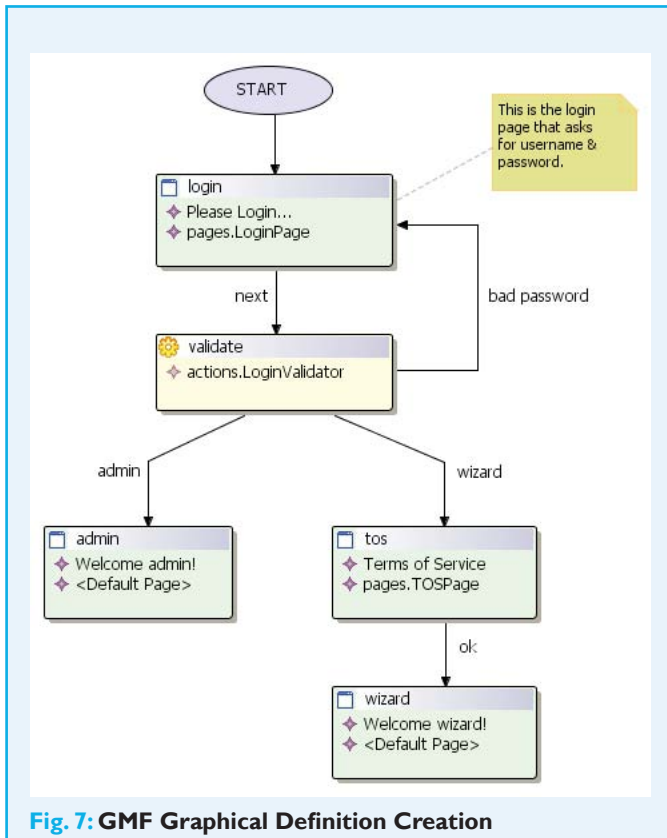


Fig. 7: GMF Graphical Definition Creation

Listing 1

Code for Drawing Custom Figures

```

protected void drawPanel(Graphics g, int delta, Image image,
                        Color panelColor, Color gradientCol) {
    Rectangle bounds = getBounds();

    // drop shadow
    g.setBackground(
        Display.getDefault().
        getSystemColor(SWT.COLOR_
WIDGET_NORMAL_SHADOW));
    g.setForeground(
        Display.getDefault().
        getSystemColor(SWT.COLOR_
WIDGET_NORMAL_SHADOW));
    g.fillRoundRectangle(
        new Rectangle(bounds.x+delta,
        bounds.y+delta,
                                bounds.width-delta,
        bounds.height-delta
                                ), 5, 5);

    // draw the panel.
    g.setForeground(
        Display.getDefault().
        getSystemColor(SWT.COLOR_
BLACK));
    g.setBackground(panelColor);
    g.fillRoundRectangle(
        new Rectangle(bounds.x, bounds.y,
        bounds.
        width-delta, bounds.height-delta
        ), 5, 5);

    g.drawRoundRectangle(
        new Rectangle(bounds.x, bounds.y,
        bounds.
        width-delta, bounds.height-delta
        ), 5, 5);

    // draw the icon.
    g.drawImage(image, bounds.x+2, bounds.y+2);

    // draw the gradient.
    g.setBackground(gradientCol);
    g.setForeground(
        Display.getDefault().
        getSystemColor(SWT.COLOR_
WHITE));
    g.fillGradient(bounds.x+18, getBounds().y+2,
        bounds.width-(20+delta), 16, false);
}
  
```

Figure Definitions

Figure 7 shows the different figures involved (except the transition figure, which is a regular polyline). The start state is modeled as an ellipse with word 'START' inside it. The display and action states are much more involved and hence need to be modeled as custom figures.

The custom figures extend `org.eclipse.draw2d.Figure` and are defined under the `custom/` directory of the 'wizard' plug-in. The labels displayed in these figures are also figures themselves. The custom figure just needs to draw the template, on top of which the child figures would be displayed. **Listing 1** shows the code for the base class (`BaseFigure.java`) that draws the template. First it draws the drop shadow and then the panel. The icon and gradient are drawn next. Also note that we are using the XY layout, which may not be the best choice, for the sake of simplicity.

GMF Tooling Model (gmftool)

We will use the GMF tooling model to add creation tools to the palette. We need four creation tools, for display state, action state, start state and transition respectively (as seen in **Figure 8**). In the 'wizard' plug-in, the file 'model/wizard.gmftool' contains the tooling definitions. To create this file, follow a process similar to that for creating gmfgraph file. Select `New` → `Other` → `Graphical Modeling Framework` → `GMFTool Simple Model`. Select 'wizard.ecore' for domain model.

In the tooling definition page, select 'Wizard' as the diagram element and check only the four elements we discussed earlier; this is shown in **Figure 9**. It will create tooling definitions with default icons. However, I have replaced the small icons with the icons from the `icons/` directory in the 'wizard' bundle.

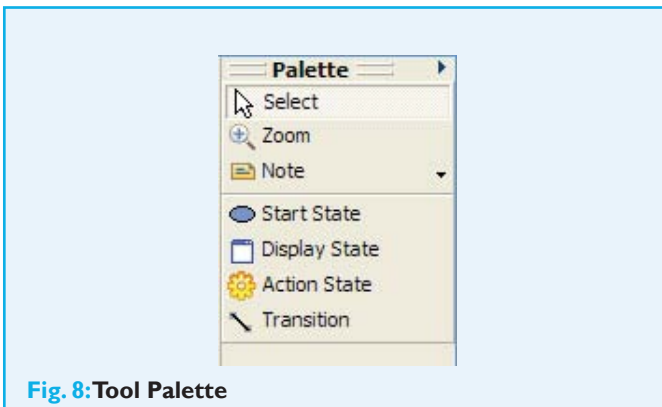


Fig. 8: Tool Palette

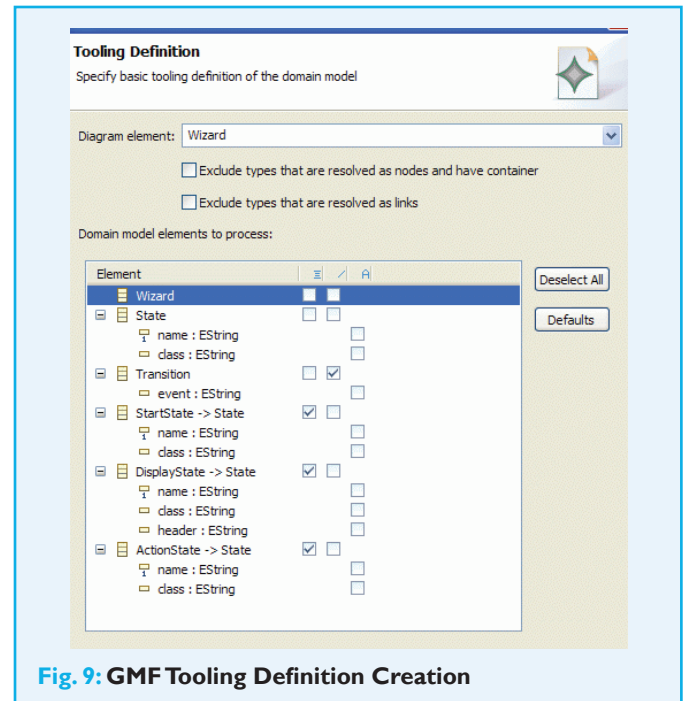


Fig. 9: GMF Tooling Definition Creation

In the tooling definition page, select 'Wizard' as the diagram element and check only the four elements we discussed earlier; this is shown in **Figure 9**.

Putting it All Together: The GMF Mapping Model (gmfmap)

As we saw earlier, the mapping model ties the domain, graphical, and tooling models together. The file `wizard/model/wizard.gmfmap` contains the mapping model. You can select `New` → `Other` → `Graphical Modeling Framework` → `Guide GMFMap Creation` to initiate a wizard that lets you create this file. Open the `gmfmap` file and observe the mappings in the property sheet.

There are three 'Top Node Reference' elements under the 'Mapping' element, corresponding to the three type of states. Under these elements are the 'Node Mapping' elements. If you click on the one corresponding to 'DisplayState', you can see that the 'Element' is mapped to the.ecore class 'DisplayState', the diagram node is mapped to the 'Node DisplayState' and the tool is mapped to 'Creation Tool Display State', as shown in **Figure 10**.

One other mapping worth mentioning is the Link Mapping. You can see that the source and target features are mapped to

‘from’ and ‘to’ EReference respectively. This will enable it to connect any two states in the diagram.

Code Generation

Now that the mapping is done, we are ready to generate code. The first step is to create the generator model. Right click on the gmmap file and select Create Generator Model.... Click Yes to accept the use of IMapMode. It will create the wizard.gmfgen file. Now right click on this file and select Generate diagram code. It will create all the necessary code and plug-ins for the graphical editor in the wizard.diagram plug-in.

To make changes to the generated code, remove the @generated tag in the method. Those methods will be skipped when you run the generator again. In fact, I had to make a couple of tweaks to the editor code. The refreshBounds() method in DisplayStateNameEditPart.java is made to force the layout of the state name figure at (25,2). This is because I couldn't get it to work using the XYLayout in the model definition. If it is not already supported, this might be addressed in future GMF releases. You can find this tweak in all the children edit parts corresponding to our custom figures.

Running the Graphical Editor

Finally we have come to the most fun part—running the editor and seeing it in action. Create a new run configuration as an Eclipse application and run it with default settings. In the target workspace, import the plug-ins wizard and wizard.framework as links. Also import the wizard.example project that is supplied.

To create a new wizard, create a Java project first. Select New → Other → Examples → Wizard Diagram. Supply a file name, say test.wizard. It will also create another file test.wizard_diagram and open it with our newly generated editor. You can add the different states and transitions between them. If you don't specify a class name for the page, the default page class wizard.framework.DefaultPage is used. Similarly for the action, wizard.framework.DefaultAction will be used. To run the wizard, simply right click on the wizard file and select the 'Run Wizard' option. Make sure you have a transition to a page from the start state.

Opening the supplied example wizard test2.wizard will produce the diagram as shown in Figure 11. Running it will bring up the pages as shown in Figure 12. Try

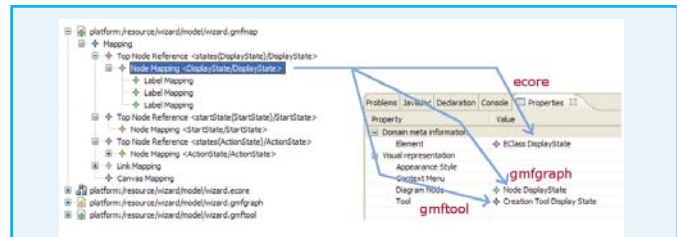


Fig. 10: GMF Mapping for DisplayState

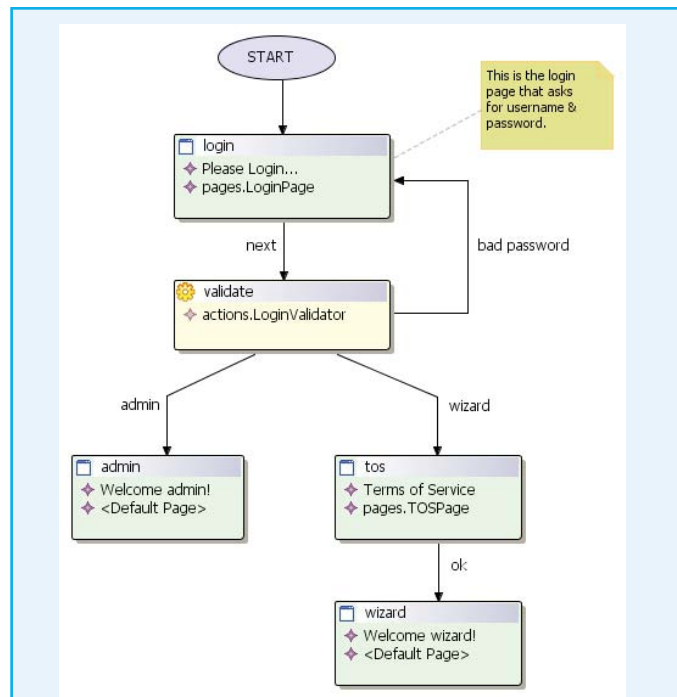


Fig. 11: Actual export of the wizard diagram “test2.wizard” as a PNG file. Right click on the diagram and select “File → Save as Image File..” to generate this.

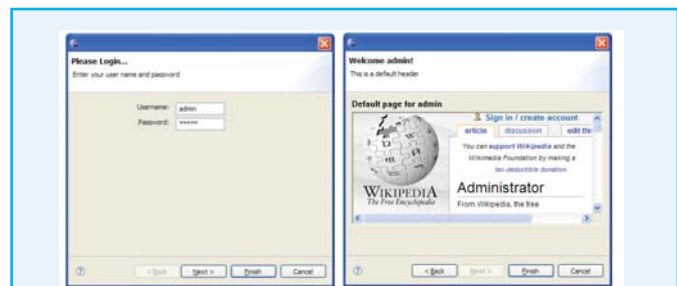


Fig. 12: Screenshots of the dynamic wizard in action.

Feature Dynamic Wizard Modeling with GMF

admin/admin as the username and password to go to the administration page and *wizard/wizard* to go to the wizard page. You can experiment by changing the page titles and rewiring the transitions and running the wizard again.

Conclusion

In this article, we went through the process of creating a dynamic wizard framework and a graphical editor using GMF from the scratch. GMF opens up endless possibilities in creating model driven graphical editors. It does, however, have a steep learning curve. Here are some tips you can put to use while working with GMF:

- Experiment a lot.
- Look at the underlying XML produced by each of the models. This will give you a deeper understanding of what's under the covers.
- Make liberal use of the Validate option found in most model editors (Right click on the root node to get that option). This can save you a lot of time.

Happy GMF programming!

Rajkumar Madhuram



is the Vice President of Engineering at Impiger Technologies in Chennai, India. Raj has been developing software for 17 years and holds a M.S in Computer Science from the University of Central Florida, Orlando.

He is an Eclipse enthusiast, and was also the winner of the International Challenge for Eclipse contest held in 2003 for his plug-in FireAntz.

Resources & References

- [1] R.Hudson, Eclipsecon 2004 presentation: http://www.eclipsecon.org/2004/EclipseCon_2004_TechnicalTrackPresentations/47_Hudson.pdf
- [2] Eclipse Wiki GefDescription, <http://eclipsewiki.editme.com/GefDescription>
- [3] Catherine Griffin, Using EMF: <http://www.eclipse.org/articles/Article-Using%20EMF/using-emf.html>
- [4] GMF Tutorial: http://wiki.eclipse.org/index.php/GMF_Tutorial
- [5] Doina Klinger, Creating JFace Wizards: <http://www.eclipse.org/articles/Article-JFace%20Wizards/wizardArticle.html>
- [6] Wikipedia article on FSM: http://en.wikipedia.org/wiki/Finite_state_machine



Every hour,
every day
round
the year...

SDA ASIA
SOFTWARE, DEVELOPMENT & IT ARCHITECTURE

Flexibility at the Roots of Eclipse

Solving the GUI Dilemma: SWT Swing and Eclipse on Swing

By Dieter Krachtus and Christopher Deckers

No trench in the world of Java is deeper than that between SWT and Swing or Eclipse and Sun. Unity is only found in the knowledge that everybody suffers from this argument. But how to end this almost religious battle over the righteous GUI-toolkit? How to bang their heads together if they only know one point of view—for them or against them! The sister projects SWT Swing and Eclipse on Swing (EOS) achieve this trick. They offer both wranglers a conciliative solution, which combines the best of both worlds and seemingly has only advantages for everybody.



Introduction

The worst choice is to have no choice at all. Therefore, it seems less a curse than a blessing to have this choice with SWT and Swing. However, the lack of time and motivation is often the reason not to master both of the GUI toolkits. Thus, it is not astonishing that the choice of the toolkit isn't focused to the current problem at hand but follows the taste of the developer, which means there was no real choice from the beginning.

Wouldn't it be great, if one had the choice between SWT and Swing not only at the beginning of a project but throughout development? How about using familiar APIs to develop Eclipse-Plug-ins, RCP-applications or a JFace/SWT GUI and still keep the option to switch back and forth between SWT and Swing without changes to your code?

SWT Swing offers this possibility, and **Eclipse on Swing** (EOS) is the proof that it even works for the most complex SWT applications, namely Eclipse itself.

SWT Swing

In August 2005 the SWT Swing project was started but it took over a year until its first official release. The SWT-Snippets^[3] are used as developer tests and to document the progress of

This article would be of interest to Technical Managers and Architects, Project Managers and Leads, and Developers. Since this paper is on interoperability, an exposure to multiple technologies (especially SWT, Swing, Eclipse RCP and Plug-ins) is desirable.

AWT, Swing, SWT, JFace, Eclipse RCP

Before taking a closer look at the projects SWTSwing and EOS, it is helpful to remember the capabilities of the solutions used in Java GUI-toolkit development.

- The Abstract Windows Toolkit (AWT) offers the developer only a meagre selection of native widgets (GUI-components like Buttons) to build his GUI. This weakness is due to the lowest-common denominator (LCD) design concept from Sun; that is, only widgets that exist on all Java-platforms were used for the implementation of AWT.
- In contrast, Swing is 100% implemented in Java and consequently very portable. This advantage, since basically all widgets are emulated, is seen by some as a weakness when it comes to performance. Similarly, the very high flexibility of Swing was achieved at the cost of an often criticised [1] complexity.
- SWT is conceptually similar to AWT, as it uses mainly native widgets. However, emulation is still possible and actually has to be used in order to realize widgets that don't exist on a certain platform. Once exclusively bundled with Eclipse, SWT is now available separately to develop standalone applications. This process has become even more convenient by using the JFace library or the Eclipse RCP.

A more verbose overview about all three UI-toolkits, also touching things like event-handling, is detailed in an IBM developerWorks article, SWT, Swing or AWT: Which is right for you? [4].

the project. These snippets are minimal examples for SWT beginners who want to learn how to build SWT widgets. SWTSwing can execute the majority of these snippets without or only minor bugs by using Swing instead of SWT widgets.

It became obvious that more complex applications and their GUIs were not only more than the sum of their pieces (widgets) but that they also show more bugs than would be expected from their pieces alone when run on Swing.

The sister-project, Eclipse on Swing (EOS) [5], was founded to serve as *Gold Standard* [9] of a complex SWT application for the development of SWTSwing. At the beginning EOS was based on a branched and heavily modified version of the SWTSwing code base, solely to convince Eclipse to start up successfully using Swing.

However, the strategy to use both simple (Snippets) and very complex tests (EOS) was very successful in speeding things up. Error-prone implementations and bugs became visible through the EOS project and led to feedback and changes in SWTSwing.

By now, these improvements of SWTSwing allow the execution of other complex SWT-applications which soon may be used in productive environments. At the moment, a few minor bugs still spoil the overall picture, which however may be already removed when this article is published. To get an impression of the current status of SWTSwing, the most

recent screenshots of the popular SWT-applications Azureus [6] and RSSOwl [7] (Figure 2 and 3) speak for themselves.

SWT(Swing) Details

Next, let's have a quick look at the general design of SWT in order to understand how SWTSwing is implemented through Swing. SWT offers a public API which has a private interface to the system by using a very thin native layer (see Figure 3).

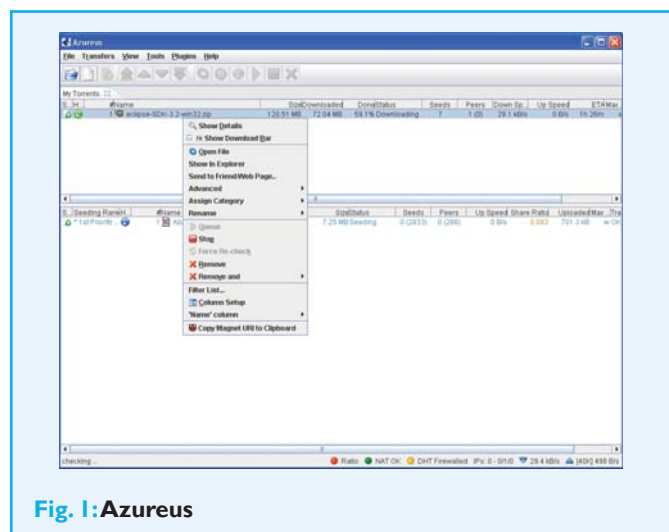


Fig. 1: Azureus

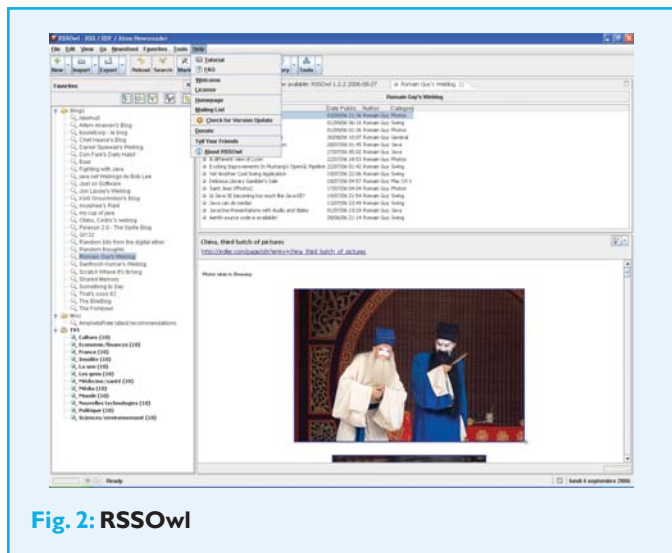


Fig. 2: RSSOwl

The actual implementation of SWT follows a general pattern. The basic idea is that all native calls are accessed through a single class called `org.eclipse.swt.internal.<given-platform>.OS`, which is a one-to-one mapping of the related C-functions. The functions of this very thin layer are directly called by the classes of the SWT public API (see **Listing 1**). Note that AWT also uses a native layer but places some logic inside making it a fatter layer compared to SWT.

In the next step we show how SWTswing actually creates Swing instead of SWT-widgets. **Listing 1** and **2** show the creation of a Button with SWT and SWTswing, respectively. The most striking difference for SWTswing in **Listing 2** is the lack of the class `org.eclipse.swt.internal.<given-platform>.OS` and thus all native dependencies. Instead the interface `org.eclipse.swt.internal.swing.CButton` allows creation of instances of the classes `CButtonPush`, `CButtonToggle` and `CButtonCheck`. These classes are specializations of the Swing-classes `JButton`, `JToggleButton` and `JCheckBox`.

For each SWT widget, there exists an interface similar to `CButton` and classes like `CButtonPush`, which implements this interface and are derived from Swing widget classes. The core of SWTswing was technically most difficult to get right, due to a fundamental conceptual difference—**event pumping** is performed explicitly in SWT, whereas Swing manages the event pump internally. To solve that problem, SWTswing defines two modes of functioning called **best-effort dispatching** and **real dispatching**.

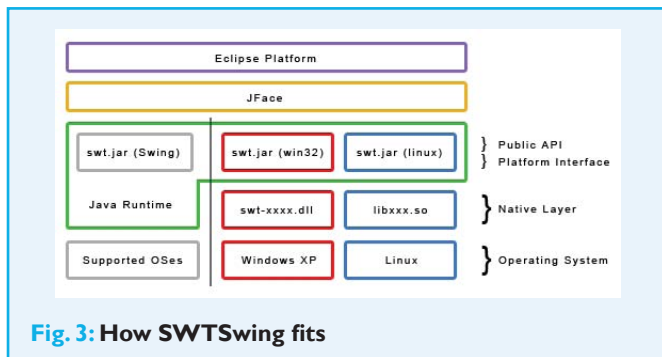


Fig. 3: How SWTswing fits

The real dispatching is the expected SWT behavior, where the SWT UI thread is the operating system UI thread. In case of Swing, that means SWT controls the automatically-managed UI thread. This is possible only if the application or the UI Thread is launched in a certain way, so it is not the default mode, but is recommended for theoretical performance gains and strict compliance with SWT's design.

The best-effort dispatching, which is the default mode, does not control the Swing-UI-thread. It consists of having two UI threads, the SWT and the Swing UI thread, both considered as valid threads for SWT calls. The

Listing 1

```
//Widget-creation in SWT for Windows:

class org.eclipse.swt.widgets.Button extends Control {
    public void setText (String string) {
        // ...
        TCHAR buffer = new TCHAR (getCodePage (), text, true);
        OS.SetWindowText (handle, buffer);
        // ...
    }
}

class org.eclipse.swt.widgets.Control {
    void createHandle () {
        // ...
        // Creation of the handle is done through the super-class
        handle = OS.CreateWindowEx (...);
        // ...
    }
}

class org.eclipse.swt.internal.win32.OS {
    // Native Methods
}
```

Listing 2

```
//Widget-creation in SWTSwing:

class org.eclipse.swt.widgets.Button extends Control {
    Container createHandle () {
        // Creation of the handle takes place within the widget
        return (Container)CButton.Instanciator.
        createInstance(this, style);
    }
    public void setText (String string) {
        // ...
        ((CButton)handle).setText(string);
        // ...
    }
}

interface org.eclipse.swt.internal.swing.CButton {
    public static class Instanciator {
        public static CButton createInstance(Button button, int
        style) {
            if((style & SWT.PUSH) != 0) {
                return new CButtonPush(button, style);
            }
            if((style & (SWT.CHECK)) != 0) {
                return new CButtonCheck(button, style);
            }
            if((style & (SWT.TOGGLE)) != 0) {
                return new CButtonToggle(button, style);
            }
            // ...
        }
    }
    public void setText(String text);
}

class CButtonPush extends JButton implements CButton
{...}
class CButtonToggle extends JToggleButton implements
CButton {...}
class CButtonCheck extends JCheckBox implements
CButton {...}
```

assumption is that most of the code executed from the SWT thread is some initialization work performed before a Swing window is actually shown and that the rest of the work is performed in response to Swing events. Under that assumption, SWT can act on Swing directly without the risk of an event originating from Swing.

Advantages

One obvious advantage of the Swing implementation of SWT is portability. Separate native libraries are not necessary, which is considered very important by some developers. Moreover, the number of supported platforms increases by those who are exclusively supported by Swing.

The look and feel support also plays a big part, when it comes to ‘company branding’. With SWT one is constrained to the native look and feel of the target platform, which in some cases is not what a developer or its company wants.

Another important point for some applications is an easy and flexible deployment. A standalone native SWT application could be deployed with a platform-specific installer, and at the same time without any code changes, as a slender platform-independent application via Webstart.

These are only a few examples where actual problems are solved because you keep the choice of using either SWT or Swing at the same time without any additional work or expense.

Eclipse on Swing

Eclipse on Swing had three motivations for its creation. One role, as mentioned earlier, was to serve as a ‘gold standard’ for the maturing of SWTSwing. The separation of the standalone code bases of SWTSwing and EOS was reverted recently after a long process of feedback and improvements, whereby now the most up-to-date SWTSwing can be used as a basis for EOS.

The EOS project provides a plug-in that hooks into the Eclipse Preferences and allows a smooth switch between using Swing and SWT. As shown in [Figure 4](#), when using Swing one can naturally select one of the various **Swing Look & Feels** that already come with the plug-in.

It is important to note, that all **Views of EOS** are already usable; only a few minor problems are left that cloud the overall picture during productive work. In [Figure 4](#), Eclipse runs with the Swing Windows Look & Feel, showing hardly any recognizable visual differences to SWT or any other native Windows application.

Another positive feature is the possibility, not only to run Eclipse itself on Swing, but in principle any application based on the Eclipse Rich Client Platform (RCP). [Figure 5](#) shows this capability of the EOS-Plug-in for the JAX 2006 Innovation Award ^[11] winner Bioclipse ^[12].

The founders of EOS and SWTSwing currently focus all efforts on improving SWTSwing. The EOS plug-in will be offered in a manually installable form, until a satisfying maturity of SWTSwing and EOS is reached. The plug-in can be downloaded from the EOS project page ^[5].

With the EOS plug-in the second goal of the EOS project is pushed forward—to provide a non-invasive plug-in to

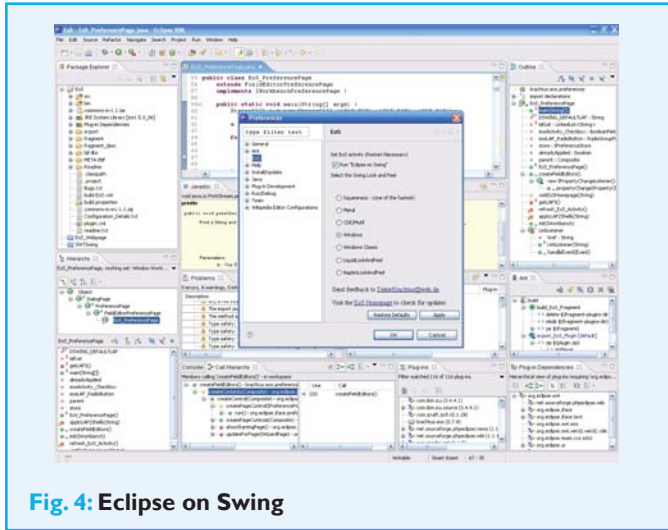


Fig. 4: Eclipse on Swing

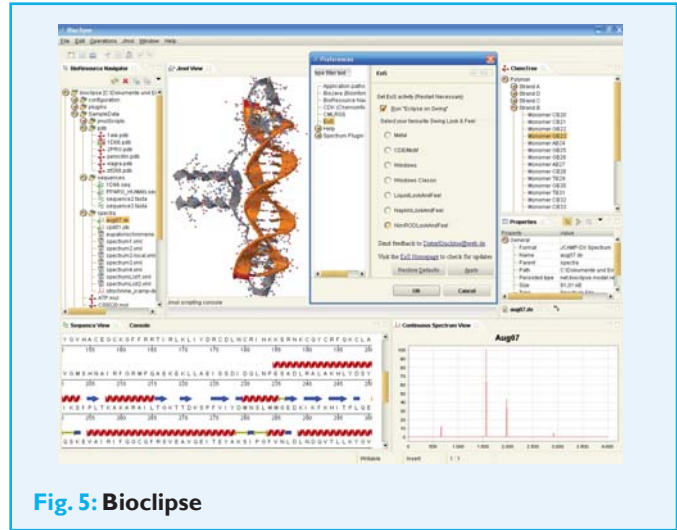


Fig. 5: Bioclipse

the Eclipse user, which can be updated from time to time to evaluate and test the progress of SWT/Swing and EOS. It could be the key element in the Java community, which mediates between SWT and Swing or Eclipse and Sun, by creating unity and destroying prejudice.

In an interview [8] Mike Milinkovich, the Executive Director of the Eclipse Foundation said: *“One of the nice things about Eclipse is that we are not religious but pragmatic. According to the motto: The people can solve their technical difficulties with this technology. It fits, it functions, it scales. It works like a charm”*.

In the spirit of Eclipse it is now possible to embrace the Swing technology (Figure 4 and 5)

The third, and in a long-term perspective, perhaps the most important motivation for the EOS project is probably not obvious when described with the words—more security and flexibility for the future of Eclipse.

Like a stock market share, the success of Eclipse strongly depends on the fact that many people believe in the future of Eclipse and rely on the promise that Eclipse will still be there in 10 years leading the bleeding edge of technology.

EOS: Eclipse on Spare (Tyre)

Accordingly, it is interesting to examine what are the most fragile parts of Eclipse, in case development has to be carried on by third parties. Firstly, there are many plug-ins for Eclipse with redundancy—meaning that free or commercial plug-ins exist that do more or less the same and which could serve as a replacement. Secondly, Eclipse is almost completely

written in Java, well documented, has Unit Tests, and so on. Besides, many developers would be more than qualified to carry on the torch of Eclipse.

The one and only element at the very root of Eclipse, that doesn’t fulfill both criteria is **SWT**. Given a worst case scenario there is neither a direct replacement exists for **SWT**, nor is the development trivial due to the native dependencies.

Even in real-case scenarios SWT/Swing/EOS could save the day; for example, to port SWT to a new version of an operating system. Swing could deliver a permanent or temporary solution, until the native SWT-implementation has reached a certain stage of maturity.

In the end, it is also less than certain that the decision to go the platform-dependent, native way will be the best solution for Eclipse. At the moment Swing is no better choice for Eclipse than SWT. In ancient times, as an aspirant for the GUI-throne, Swing was once scolded as ‘Prince Valium’ not without reason. However things turned for the better when it came of age. So who can tell what improvements for Swing a JRE 10.0 has to offer?

Summary

Accepted common sense—namely to choose the technology that solves a problem most elegantly—is usually ignored when it comes to Java GUI development. Rather, once the decision for a GUI toolkit is already cast, arguments and justifications for why this is the most pragmatic decision are found afterwards. The founders of SWT/Swing/EOS and

Cover Story Flexibility at the Roots of Eclipse

authors of this article stick with Orwell, who reminds us that you usually cannot choose between good and evil but rather the lesser of the two evils.

This means two things. Firstly, there isn't an ultimately best toolkit—SWT and Swing both have their weaknesses. Secondly, from now on, you are in the lucky position to choose between one of these two evils throughout development.

Feedback on this article can be mailed to editors@eclipsemag.net.

Resources & References

- [1]
- [2] IBM developerWorks: www-128.ibm.com/developerworks/opensource/library/os-swingswt/ca=dgr-lnxw01WhichGUI
- [3] SWT-Snippets: www.eclipse.org/swt/snippets/
- [4] SWT-Swing: swtswing.sourceforge.net
- [5] Eclipse on Swing: eos.sourceforge.net
- [6] Azureus: azureus.sourceforge.net
- [7] RSSOwl: www.rssowl.org
- [8] The Executive Director of the Eclipse Foundation Mike Milinkovich, in Eclipse Magazin Vol. 8
- [9] en.wikipedia.org/wiki/Eos
- [10] en.wikipedia.org/wiki/Gold_standard_%28test%29
- [11] http://jax-award.de/jax_award/index_eng.php
- [12] www.bioclipse.net

Dieter Krachtus



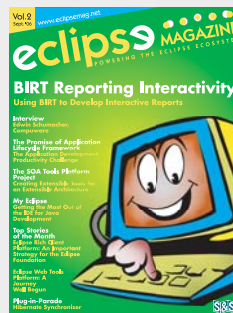
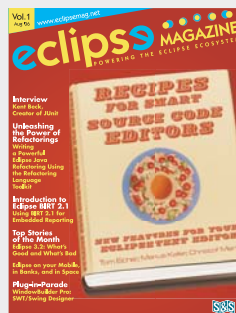
is the founder of Eclipse on Swing (EOS) and co-developer of SWT-Swing. He works as a developer and consultant with a main focus on Rich Clients on the basis of Swing or the Eclipse RCP. Currently he does a PHD at the Interdisciplinary Center for Scientific Computing (IWR) in Heidelberg.

Christopher Deckers



is the founder of SWT-Swing and co-developer of EOS. He lives in France and works as a Senior Java developer. Christopher had always an interest in rich user interfaces, tools and APIs and has his share in many Open source projects.

Information Power for the Eclipse Ecosystem



Download All Issues of the Magazine for FREE.
Visit www.eclipsemag.net



- Fresh, cutting-edge information on Eclipse
- Content written by industry experts and peers
- Published monthly, in an exclusive digital format
- For daily news updates & feature stories, log on www.eclipsemag.net

Subversive

The Eclipse Plug-In for Subversion

By Frank Schröder

Version control systems play a central role in software engineering. In the beginning, CVS was the poster child versioning system of the open source community. Then Subversion was developed because many felt CVS could not keep pace with changing technologies and practices. New plugins such as Subversive and Subclipse have appeared in the Eclipse ecosystem to connect Eclipse developers and Subversion. The article introduces you to Subversive.



Introduction

In the recent past, version control systems came to play a central role in software engineering. With the proliferation of distributed global development, they have become indispensable. It is vital to both large and small teams to be able to access the actual state of the development, and highly desirable to have access to changes throughout the history of any given project. In recent years, CVS became the poster child versioning system of the open source community.

However, many felt that CVS could not keep pace with changing technologies and practices. This led to the development and release of **Subversion** (SVN) in 2001, which many now view as the successor to CVS. Originally developed by the ‘fathers of CVS’ Ben Collins-Sussman and Karl Fogel, SVN inherits its status as an open source system. However, its main focus is to eliminate the weak points of CVS that could not be practically remedied without a fresh start.

For example, SVN has support for refactoring a concept that was unknown when CVS was originally designed—handling the full file history.

CVS became the de facto standard of the open source world, as can be seen by the fact that it comes with Eclipse version 3.2. However, the winds of change are blowing, and Subversion looms large on the horizon of interest. New plugins such as **Subversive** and **Subclipse** have appeared in the Eclipse ecosystem to connect Eclipse developers and Subversion. The rest of this article will introduce **Subversive**.

Subversive Overview

Subversive is developed by Polarion Software, an international vendor of Application Lifecycle Management (ALM) solutions including *Polarion for Subversion*^[1], which handles Requirements, Change, Project, Task, and

Build Management for all sizes of software development organizations. Subversive is based on Subversion and was originally developed as one of the clients for the Polarion Server.

In March of 2006, Subversive became open source software and was offered to the Eclipse community where it rapidly became one of the most popular Eclipse plugins (see Eclipse Plugin Central – EPIC).

Subversive is incorporated into FastTrack—a free plugin for Eclipse that provides issue tracking and management features in Eclipse in addition to providing as a Subversion client interface via Subversive. FastTrack is available as a free download at <http://www.polarion.com/fasttrack/> (registration is requested). FastTrack with Subversive is available for Eclipse 3.0 and SVN 1.1 or higher – the Java SDK 1.4 or 1.5 is also needed.

FastTrack is a FREE tracker plug-in for Eclipse. It provides basic issue tracking, plus basic project planning and team collaboration, all deeply integrated into the Eclipse IDE. For more information on FastTrack read Volume 5 of Eclipse Magazine.

Getting Started with Subversive

Assuming an existing installation of the Eclipse IDE and Subversion, you can use the Eclipse Update Manager to get the latest version^[2] of Subversive.

Subversive adds the Repository Exploring perspective to the Eclipse IDE where you find the same level of support for Subversion as Eclipse offers for CVS. You can use the client interface to define ‘Repository Locations’ that access any root in your SVN repository. Using the Subversive client in the Repository perspective you can:

- Browse the repository and modify the resources (Files)
- Checkout
- Synchronize
- Commit
- Update
- Solve conflicts
- Merge

Subversive’s Focus

The Subversive development team placed much emphasis on the user interface and general usability of the plugin,

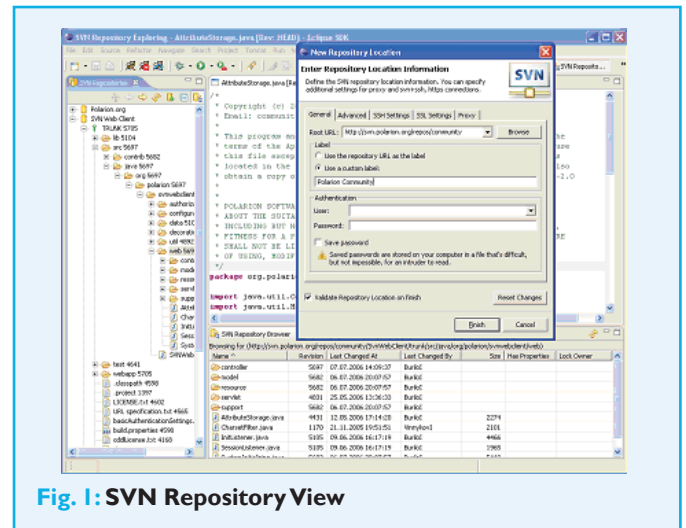


Fig. 1: SVN Repository View

incorporating a great deal of feedback from the community all during the development cycle. Some of the post popular features^[3] include:

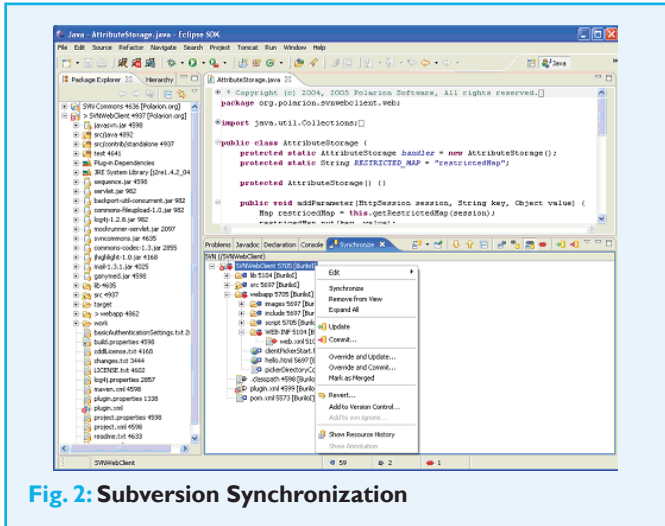
- Cross project atomic commits
- Support of recommended repository layout, including trunk, branches and tags layout
- Recursive directory revision comparison
- Adding repository locations with same URL validation in forms
- Extended commit dialog

Subversion is based on libraries and executables developed in the C programming language. To connect it to a Java based environment like Eclipse, it is necessary to use special Java interfaces. There are two libraries which support this connection: JavaHL and JavaSVN. Subversive supports both, although there is extended functionality in JavaSVN because both development teams have extended the Subversion API. Subversive supports the following features for JavaSVN:

- Interactive merge operation, similar to merge in the Eclipse CVS plug-in
- Full projection of Eclipse refactoring operations into Subversion
- Support of recommended repository layout, including trunk, branches and tags layout
- Automatic resolving of conflicts

Subversive – Next Steps

The Subversive projects seeks to become the standard way that Eclipse connects with Subversion in the same way that 'Eclipse CVS Team Project' enables connectivity and ease of use with CVS. In the summer of 2006, Polarion Software submitted Subversive as an Eclipse proposal [4] to become the Eclipse/SVN connector included in the Eclipse distribution.



Resources & References

- [1] <http://www.polarion.com/sub/>
- [2] Download the latest version: <http://www.polarion.org/index.php?page=download&project=fasttrack>
- [3] Subversive Features: <http://www.polarion.org/index.php?page=features&project=subversive>
- [4] Subversive Project Proposal: <http://www.eclipse.org/proposals/subversive/>
- [5] www.polarion.org
- [6] <http://www.eclipse.org/proposals/>
- [7] www.tmate.org/svn
- [8] www.svnbook.red-bean.com
- [9] http://www.polarion.com/sub/index_pro.php
- [10] www.tigris.org

Frank Schröder



is CEO of Polarion Software. At Polarion Frank is focused on building a market-driven company for software development platforms. He brings more than 16 years of experience at senior levels in IT/software companies. He has built international operations and managed Merger and Acquisitions.

IT'S FRESH!



The all new

www.sda-india.com

The bright, new face of Enterprise & IT Management.

Deploying the BIRT Viewer to JBoss

Disseminate Report Content to an Application Server

By Jason Weathersby

With information applications, developing content for delivery is only a part of the equation. After report designs are complete, the infrastructure for deploying the application has to be addressed. This is equally true for BIRT applications. In previous articles, we discussed many of the BIRT Designer's features, but in this one we will cover deployment options available to the BIRT developer, with an emphasis on deploying the Example BIRT Viewer to the JBoss Application Server—although the techniques discussed in this article should apply to most J2EE-compliant application servers.



Introduction

The BIRT project supplies a variety of components for download. The BIRT Designer, which is an Eclipse Perspective, is used to develop report designs, templates and libraries. The Charting package provides support for complex charting and is integrated with the BIRT Designer. A stand-alone version of the Charting component is also available. The BIRT Viewer is an AJAX-based JSP/Servlet application that wraps the various BIRT APIs to deliver reports. The APIs included are the Report Engine API (RE API), Chart Engine API (CE API) and the Design Engine API (DE API). The DE API is used to create XML-based report designs, templates and libraries, and is used by the BIRT Designer. The CE API is used to design and create various chart types and is used by

all BIRT components. The RE API is used to load, run, and render XML based designs created by the BIRT Designer.

All the BIRT components and APIs offer a wide range of integration and extensibility options. This article will focus on the BIRT Viewer and its deployment to the JBoss Application Server. I will demonstrate the concepts using BIRT 2.1.1 and JBoss 4.0.5.GA. We will begin by presenting an overview of the BIRT Viewer, followed by instructions for deployment to the JBoss Application Server and conclude with details on setting up and using a JBoss supplied connection pool.

BIRT Viewer Overview

BIRT supplies an example viewer that wraps the BIRT APIs for report generation and rendering. This viewer is encapsulated

Requirements to Deploy and Run Examples

The article demonstrates the use of GMF in building a graphical editor. First a framework for JFace wizards is developed, that allows one to dynamically control the page flow. Next the author maps the domain to GMF elements and shows how to construct a graphical editor with very little effort.

in an Eclipse plug-in that the BIRT Report Designer uses to preview the reports while in development. The BIRT Viewer is also available in a separate download as a JSP/Servlet application that can be deployed to a J2EE application server. The viewer has two modes of operation that correspond to two servlet mappings (run and frameset) within the viewer application.

The frameset mapping is used to display an AJAX-based viewer that has controls for table of contents, parameter entry, export to CSV, and printing of reports. Additionally this mapping supports paginated HTML and features a set of page navigation controls.

When using the run mapping, reports are returned as PDF or HTML. With this mapping the HTML that is returned is not paginated. This mapping also does not make use of interactive features such as table of contents and export to CSV. The designer uses this mapping when the preview tab is selected in the report editor.

The viewer supports many URL parameters and application settings that determine how reports are processed. When

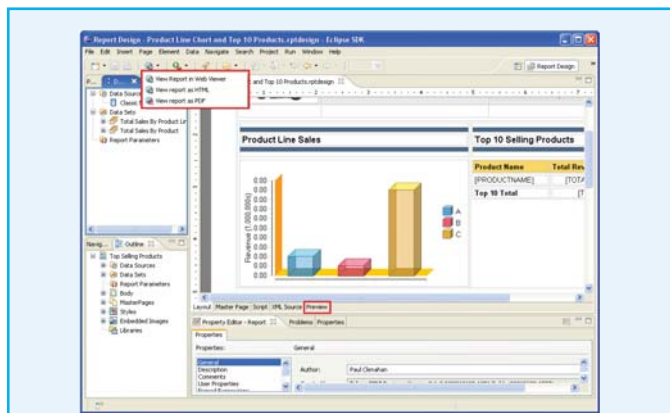


Fig. 1: Options for Launching the BIRT Viewer within the BIRT Designer

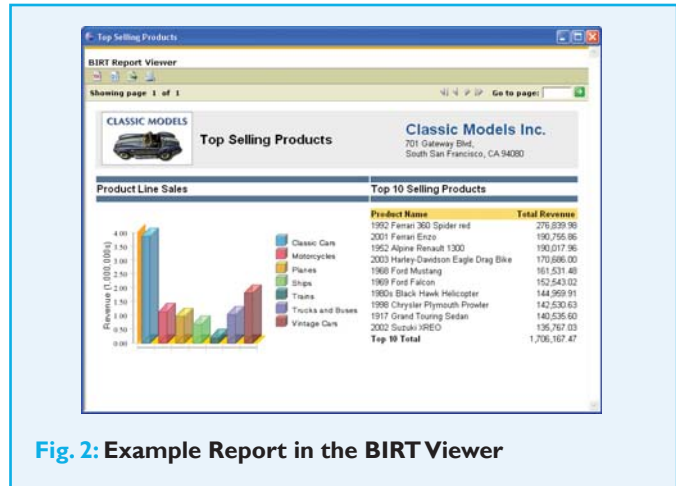


Fig. 2: Example Report in the BIRT Viewer

deploying the Viewer these settings can be very important. Some of the key parameters and settings are presented below.

The `__report` URL parameter specifies the name of the report design to run. For example, use `http://localhost:8080/birtweb/frameset?__report=c:/tmp/Sample.rptdesign` to execute a sample report located in the `tmp` directory. The path can be absolute or relative. If the path is relative, the Viewer will search for the report design relative to the context location. Relative paths also work when the Viewer is deployed as a `.war`. The only drawback is that the reports need to be contained within the `war`, unless the `BIRT_VIEWER_DOCUMENT_FOLDER` attribute is set. This attribute is discussed below.

The `__document` URL parameter specifies the name for the report document. When using the frameset mapping the Viewer divides report creation into two phases—Generation and Presentation. In generation a report design (`.rptdesign`) file is read and a report document is generated (`.rptdocument`). In the presentation phase the report document is rendered to the Viewer. This allows the AJAX code embedded in the Viewer to perform pagination and other interactive functions. The report document is automatically created by the Viewer application, unless the `__document` parameter is set. If the `__document` parameter exists, the Viewer will use it as the filename for the report document. The path can be absolute or relative. If the path is relative, the Viewer will search for the report document relative to the context location. If you specify a `__document` parameter with no `__report` parameter the Viewer can be used to render an existing report document. This may be useful when report generation is time consuming and the data changes infrequently.

Feature Deploying the BIRT Viewer to JBoss

Both the `__report` and `__document` parameters can be relative to a file location specified in the `web.xml` for the Viewer. The setting is `BIRT_VIEWER_DOCUMENT_FOLDER`.

```
<context-param>
  <param-name>BIRT_VIEWER_DOCUMENT_FOLDER</param-name>
  <param-value>c:/tmp</param-value>
</context-param>
```

Setting the `BIRT_VIEWER_DOCUMENT_FOLDER` as above will result in the `c:/tmp` directory being searched for relative values of `__report` and `__document` parameters.

The `__resourceFolder` URL parameter is used to set the resource folder for the BIRT engine. BIRT uses a resource folder to contain reusable components such as libraries and images. This parameter is expected to be an absolute path. There is also a setting in the `web.xml` for the resource folder. If neither setting has a value, resources are collected from the same folder as the report designs. This setting is very important when deploying reports that use libraries and other resources.

The `__format` URL Parameter is used to specify the output format when using the run mapping. The current values are HTML and PDF. This list will be expanded in the 2.2 release of BIRT.

The `__locale` parameter is used to set the locale for the given report. There is also a setting in the `web.xml` file to set the locale.

Report parameters are specified by `ReportParameter=ParameterValue` pairs. To set the value of `MyParameter` for the

`MyReportDesign`, the URL would be formatted similar to:

http://localhost:8080/birtweb/frameset?__report=MyReportDesign.rptdesign&MyParameter=TestMyParameter.

Customizations to the Viewer can be accomplished by checking out the `org.eclipse.birt.report.viewer` project from CVS as described on the BIRT web site. In addition to checking out the source code, customizations to the BIRT Viewer can be accomplished by making changes to the JSP fragments, JavaScript files, and Styles used by the BIRT Viewer. The directory structure for the BIRT Viewer is presented below with a description of the contents of key locations.

Deploying the Viewer to JBoss Application Server

JBoss supports many ways to build, package, and deploy applications. These methods are covered extensively on the JBoss web site. Ideally we would create an ANT project to deploy the BIRT Viewer, but for brevity we will illustrate deployment as a simple copy. The example BIRT Viewer is packaged as part of the `birt-runtime-version` download. From the BIRT web site select the 'Download' link. The options available are as shown in **Figure 4**.

The Runtime link listed under Deployment is the package that contains the example BIRT Viewer. In this article we

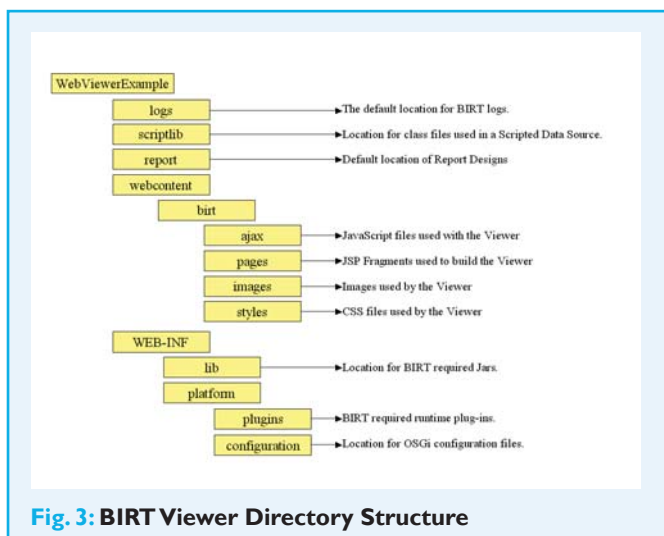


Fig. 3: BIRT Viewer Directory Structure

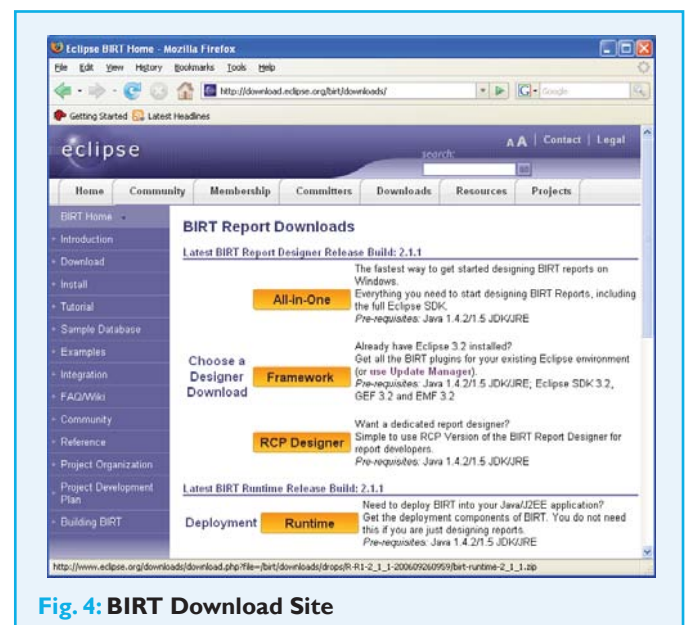


Fig. 4: BIRT Download Site

are using BIRT 2.1.1, so download the birt-runtime-2_1_1.zip file and extract the birt-runtime-2_1_1/WebViewExample directory to a temporary location. The WebViewExample folder contains everything we need to deploy to a J2EE compliant application server, with the exception of iText jars and JDBC drivers.

The iText jars are used for PDF generation and need to be added to WebViewExample. BIRT 2.1.1 uses the itext-1.3.jar, which can be downloaded from <http://prdownloads.sourceforge.net/itext/itext-1.3.jar>. If your reports need PDF Asian font support you will also need iTextAsian.jar, which can be downloaded from <http://prdownloads.sourceforge.net/itext/iTextAsian.jar>. After downloading these files, create a lib folder in the WebViewExample/WEB-INF/platform/plugins/com.lowagie.itext directory. Copy both jars to this location.

JDBC drivers are typically copied to the WebViewExample/WEB-INF/platform/plugins/org.eclipse.birt.report.data.oda.jdbc_2.1.1.v20060922-1058/drivers directory, but because we want to setup connection pooling through JBoss, we will skip this step.

To deploy to the JBoss Application Server, rename the WebViewExample directory to applicationname.war (for example, birtweb.war) and copy it to the \$JBOSS_HOME/server/your_configuration/deploy directory (for example, \$JBOSS_HOME/server/default/deploy in the default configuration). JBOSS will automatically deploy the exploded war.

Another option is to war the Viewer using the following command within the WebViewExample directory.

```
jar -cvf birtweb.war
```

After the WAR is created, copy it to the \$JBOSS_HOME/server/default/deploy directory and JBoss will automatically deploy the application.

Testing the Viewer

After deploying the Viewer, start the JBoss Application server and enter <http://localhost:8080/birtweb> as the URL and select the View Example link. The output will be similar to that shown in [Figure 5](#).

Although this report is not very exciting it verifies that the Viewer is running correctly. We can now move on to deploying user-created reports.

Deploying a Report

In most cases you will want to make changes to the file web.xml as described in the Viewer section of this article to

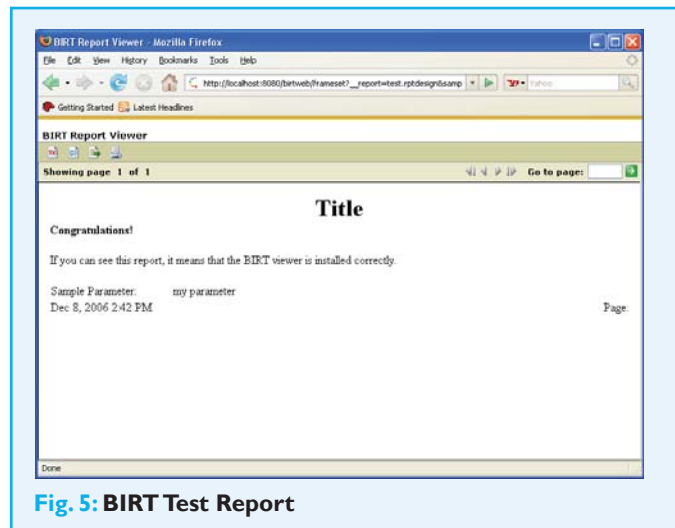


Fig. 5: BIRT Test Report

deploy reports. This will include setting values for the report document locations and resource locations. One key item for deployment that we touched on briefly in the last section is JDBC drivers. BIRT has a default directory for JDBC drivers, but in deploying to JBoss we would prefer to share a connection pool provided by the Application Server. This is fairly simple to set up with JBoss and is a logical choice for deploying BIRT reports.

In this article we will demonstrate with the MySQL Connector/J driver, which is contained in the mysql-connector-java-3.1.12-bin.jar. As per the JBoss instructions, we first copy the driver to the \$JBOSS_HOME/server/configuration/lib directory.

Next we make a copy of the mysql-ds.xml file located in the \$JBOSS_HOME/docs/examples/jca directory and make the following changes highlighted in 'bold>'. (See *Configuring JDBC DataSources* in *The JBoss 4 Application Server Guide* for more information).

```
<datasources>
<no-tx-datasource>
  <jndi-name>MySQLDS</jndi-name>
  <connection-url>jdbc:mysql://localhost:3306/Classicmodels</connection-url>
  <min-pool-size>5</min-pool-size>
  <max-pool-size>15</max-pool-size>
  <driver-class>com.mysql.jdbc.Driver</driver-class>
  <user-name>root</user-name>
  <password>root</password>
  <exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.
```

Feature Deploying the BIRT Viewer to JBoss

```
MySQLExceptionSorter</exception-sorter-class-name>
```

```
<metadata>  
  <type-mapping>mysql</type-mapping>  
</metadata>  
</no-tx-datasource>  
</datasources>
```

We are setting this data source up as a `no-tx-datasource`, because we do not require transaction support. The pool size is set to be a minimum of five and a maximum of 15 connections for example only. The driver is set to the MySQL driver and the connection URL is set to a local copy of the Classic Models database available from the BIRT web site. We will discuss building a report that uses this data source in

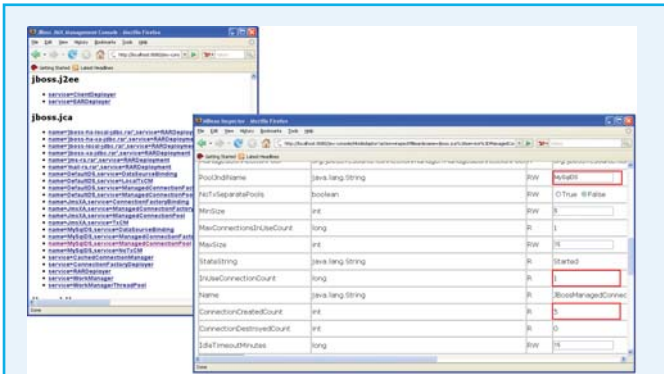


Fig. 6: JMX Management Console

the next section, but when running a report against it you can use the JMX Management Console to view the connections.

Now that the Viewer is deployed to the JBoss Application Server and the connection pool is set up, we can launch the BIRT designer and start developing reports to deploy.

Building a Connection within the BIRT Designer

Illustrated below is the BIRT Data Source Editor for a JDBC connection. This editor can be reached by adding a new data source to a report or by double clicking on an existing data source. In this example the driver class is set to the MySQL driver and the URL points to the local copy of the Classic Models database. We also have supplied a username, password, and JNDI URL. The JNDI URL is used to look up a connection from a Naming Service.

When designing a report to use in this environment, we could use JBoss' support for remote client access, but this is not recommended, and would require additional changes to the `mysql-ds.xml` file. Instead, our data source is defined under the `java:/` context, which will allow access to the pool by other applications within the same virtual machine, including the deployed BIRT Viewer, but not the BIRT Report Designer.

The BIRT JDBC Data Source, whether used within the designer or from the deployed Viewer, tries to use the JNDI name to do a lookup, passing the supplied username and password. If the lookup or connection fails, the driver

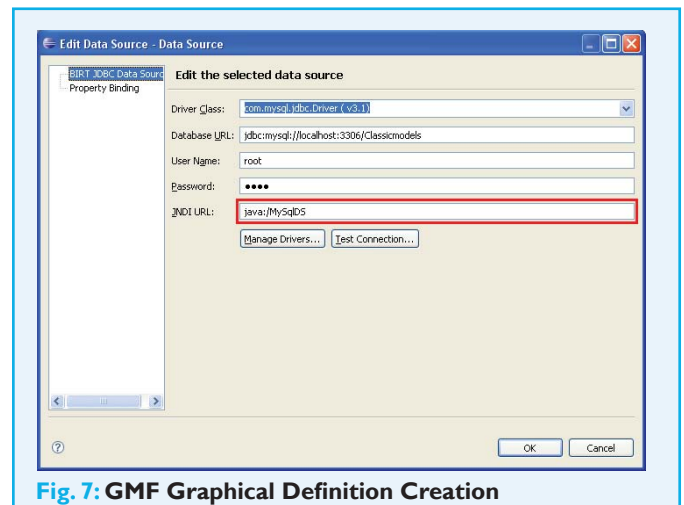


Fig. 7: GMF Graphical Definition Creation

attempts a JNDI lookup with no username and password supplied, which will default to the values we configured earlier in the `mysql-ds.xml` file. It is important that the same username and password are used for each request to the pool. If this is not the case, the pool will automatically be rebuilt when the credentials change. If both lookups fail, the BIRT driver will try to make a direct connection using the supplied Driver Class, URL, User Name and Password.

This presents somewhat of a problem when developing reports. Ideally we would have a username and password that is used for development, and use the supplied `mysql-ds.xml` credentials when the report runs in the deployed version of the Viewer. Fortunately BIRT provides a mechanism to enable this. BIRT Data Sources have a feature called Property Binding that allows public driver properties to be modified at runtime. This is accessible by selecting Property Binding entry while in the Data Source Editor.

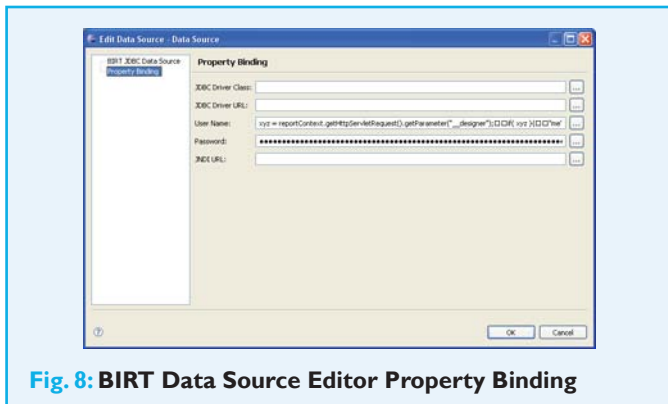


Fig. 8: BIRT Data Source Editor Property Binding

The BIRT JDBC driver supports changing the Driver Class, Driver URL, User Name, Password, or JNDI URL within the property binding feature. When a property binding value expression is entered, it will be evaluated at runtime to return a value. This can be very useful for running the same report on different databases in production and development, as well as other situations.

We can use this feature to change the username and password depending on how the report is executed. Expressions in the Data Source Property Binding editor can reference many values, such as parameters, session variables, or script values. If we enter the expression shown below for User Name and Password, the engine will use 'me' as the username and password to make a local database connection while running the report in the designer. On the other hand, the username and password will be set to blank when running in the Web Viewer deployed in JBoss, forcing the use of the connection pool credentials.

To test the property binding, we can build a report that includes



Fig. 9: Property Binding Expression

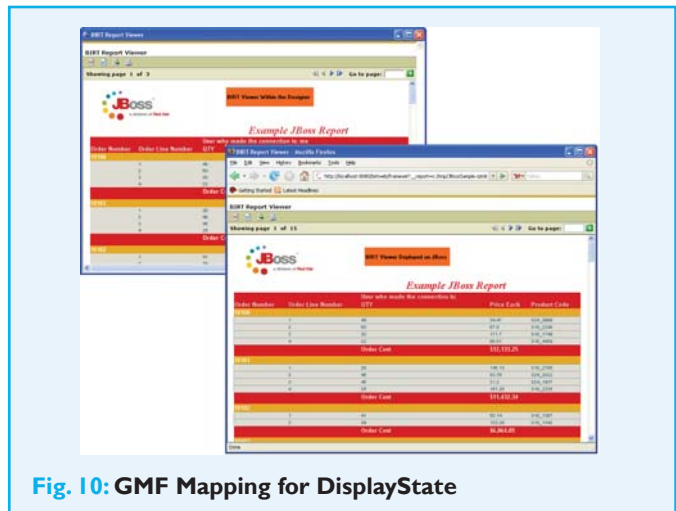


Fig. 10: GMF Mapping for DisplayState

the expression above. We can also add a field to the report that shows the user who made the connection. This value will be 'me' if run locally and '' if executed on the JBoss Application Server.

Summary

In this article we have discussed one deployment scenario available to the BIRT developer. Deploying the BIRT Viewer to an application server is only one of many options available for disseminating report content. Deploying using any of the BIRT run time APIs are valid approaches. Wrapping the BIRT Viewer or the APIs in portlet technology provides another means of integration. While there are many options available, information covered in this article should be pertinent to any deployment strategy involving J2EE application servers.

Resources & References

- [1] www.jboss.org
- [2] www.eclipse.org/birt

Jason Weathersby



is the BIRT Evangelist at Actuate Corporation and a member of the Eclipse Business Intelligence and Reporting Tools (BIRT) Project Management Committee (PMC). Jason has over 15 years experience in the soft ware development fi eld, ranging from real time process control to business intelligence soft ware. At Actuate, Jason is currently responsible for educating the Open Source community on BIRT and encouraging its adoption.

Enabling Integration and Interoperability for Eclipse-based Development

An Introduction to the Corona Project

By Edwin Shumacher

Designing, developing, testing and managing business critical applications has become increasingly more complex. To manage this complexity, projects are typically divided into tasks and teams are assigned to execute them. But IT managers also need to ensure all these different teams and team members collaborate effectively as if they were a small team all working in the same room, in the same office, and on the same project. The article explains why Corona is the right tool to address this IT business problem.



Introduction

Designing, developing, testing, and managing business critical applications has become increasingly more complex. Today's applications have a distributed architecture; they implement multiple languages and span multiple platforms from browser to mainframe. In addition applications have many integration points with packages and legacy applications.

To manage this complexity, projects are typically divided into tasks and teams are assigned to execute them. As a result, the IT organization responsible for delivering and managing these business critical applications has become increasingly complex as well and is required to interact with multiple organizations.

In order to ensure the application will deliver on the expectation of the business it is common sense to involve **business stakeholders** in the requirements capturing phase. Also, to ensure the project will be run as cost effective as possible, organizations may execute a strategy called multi-sourcing—a combination of in-house development and outsourcing to local/specialized SI's and offshore development. **Knowledge and experience** are further

Corona is a server-side framework that enables Eclipse-based tools to collaborate, sharing information about projects, applications and events. It provides Eclipse developers with a wider selection of plug-ins to more quickly diagnose and resolve application problems.

key factors in making a project successful, however the knowledge that you need may not be available in one location, but spread across different locations. From a tools perspective, most people will probably favor the Eclipse IDE, but we have to acknowledge that there are more IDE's available in the market. In addition, IDEs are used by **developers** whereas **testers** need testing tools and **business analysts** need requirements management tools and **operations** people need tools to help them monitor applications running in production.

As if this is not difficult enough the business users want IT to deliver more value and better service, more quickly and at a lower cost.

So the question is how to involve all these different teams and team members and make them collaborate effectively as if they were a small team all working in the same room in the same office on the same project? That is the IT business problem that Corona is aiming to address.

What the Market Needs

To address these challenges, large companies like Microsoft and IBM have been working on individual solutions. Microsoft provides **Visual Studio Team System** that utilizes information in the Team Foundation Server to address the needs for communication and collaboration across the lifecycle. At EclipseCon 2006, IBM gave a demo on **Jazz**. Jazz is an IBM proprietary solution for collaborating information across the development environment. There are other companies also working on collaboration solutions.

Compuware agrees with Microsoft and IBM that a solution is required to help solve these business challenges, because the existing means of communication, such as meetings, conference calls, groupware, wikis and *slashdots*, to fill these gaps are lacking and require the team to go outside their normal environment (using manual processes). We need to automate these processes so they become part of the daily activities of the team.

So why not use Microsoft or IBM solutions? In reality customers need to have a solution that's open to both Microsoft and Java based applications. By providing this solution, many different tool vendors with best of breed products will be able to participate in tool solutions for customers that share information across the full lifecycle.

What is Corona?

Corona is about tool integration. It will enable Eclipse constructs, such as a 'project', to be shared in a distributed environment. The power of the Eclipse Workbench will be extended to enable a project team to function as a workgroup.

Within the Eclipse Workbench, the project provides the context for plug-in interoperability. Plug-ins can share the same project and its resources. They can receive notifications when a project or its resources have been updated to allow them to take appropriate actions. However, this is limited to the scope of a single Eclipse Workbench instance.

See Volume 2 of Eclipse Magazine where Edwin Schumacher speaks about the Corona Project, how it works with ALF, and more.

Corona extends the knowledge of a *project* to the server-side. This allows multiple Eclipse Workbench instances to share the same project definition. Corona introduces a **Collaboration Nature** that can be attached to a project. This enables project notifications that are local to an Eclipse Workbench to be published as **Collaboration Events** and shared with all members of a workgroup. All project team members are notified when updates are made to project resources. This allows project team members to be more proactive, instead of reactive, to project events.

Eclipse = Workbench Interoperability

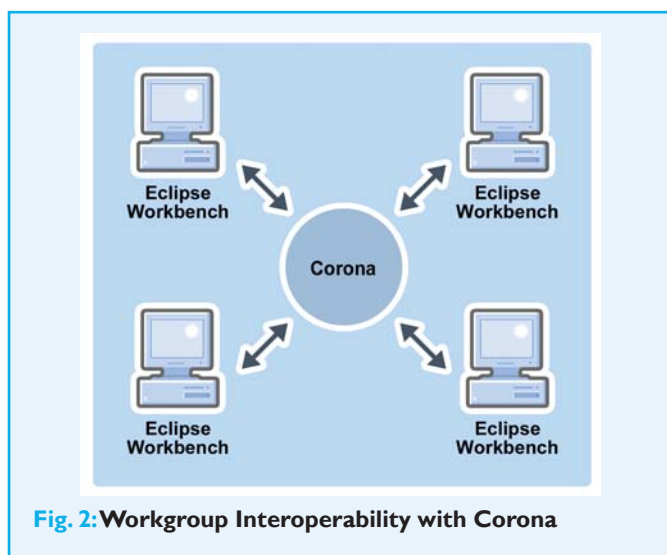
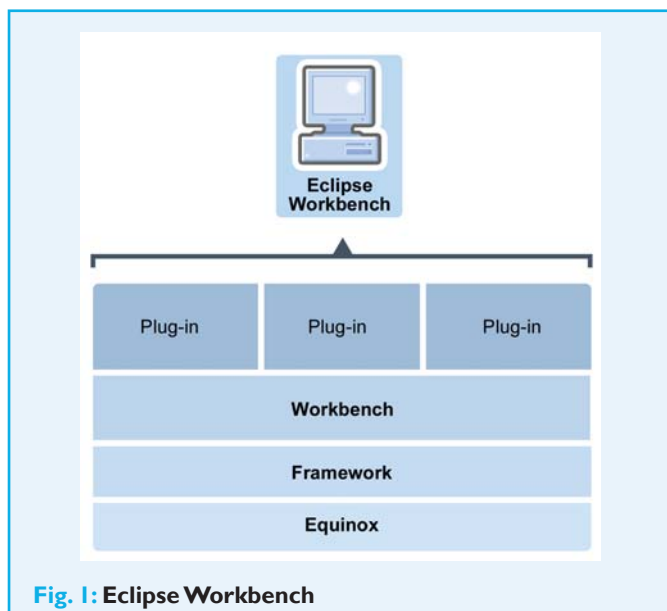
Eclipse is originally known as an IDE. It is a platform for tool vendors to provide plug-ins to empower the developer. As the popularity of Eclipse grew, it became apparent that its plug-in model could be used for applications in other vertical markets. Beginning with Eclipse version 3, the Eclipse Workbench became an independent platform separate from the IDE. The IDE became an application based on the Workbench.

The Eclipse Workbench provides the foundation for Eclipse based Rich Client Applications (RCP). It defines common constructs such as projects, resources, and events that enable interoperability across multiple plug-ins. In addition to these common constructs, the Workbench also provides technology frameworks configuration settings, installation and updates, online help, and many other capabilities that unite plug-ins into a cohesive solution. See [Figure 1](#).

Corona = Workgroup Interoperability

Corona extends the Workbench interoperability to the Workgroup. The definition of a project is moved to the Corona server so that it can be shared with multiple Eclipse Workbench applications. The common shared project definition now provides the context for collaboration across all *Workbenches*, thus enabling the Workgroup.

Project notification events that were once local to a single Workbench are now shared across the Workgroup as



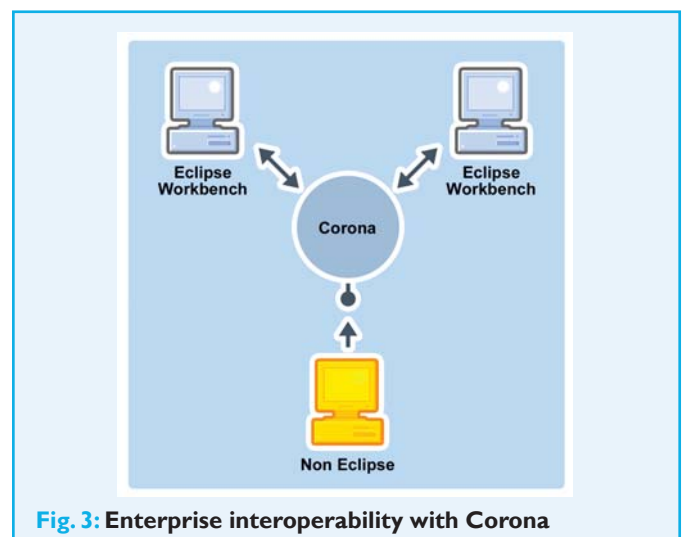
Collaboration Events. These events enable the members of the Workgroup to stay synchronized with one another.

Finally, Corona will extend the capabilities of the Eclipse Communication Framework (ECF) to provide a context to collaboration features such as chat and shared editors. See **Figure 2**.

Corona = Enterprise Interoperability

Not all tools are based on the Eclipse Workbench. Corona acknowledges and respects this and is therefore based upon a Service-Oriented Architecture (SOA) component model. Corona plug-ins can be enabled to expose functionality via Web Services.

The benefit of this design is that it will allow additional Corona plug-ins, both commercial and open source, to be deployed within Corona. These plug-ins can act as proxies to provide integration with non-Eclipse solutions. See **Figure 3**.



Example User Scenario – Production Issue Resolution

Imagine you are having problems in your production system. They are intermittent but are becoming increasingly more critical to the integrity of the system. Many team meetings between different operational areas have been held to determine the cause of the problem. A special team has been assigned to try to identify the cause of the problem and determine a resolution. This scenario shows how monitoring and debugging tools can plug into the Corona framework

Corona Frameworks

The core components of Corona are:

- **Core Framework:** Corona is based upon Equinox to provide the basis for a SOA environment. Equinox has implemented the OSGi specification. In addition to the OSGi runtime, Corona also leverages several other Eclipse frameworks. The result is a consistent Eclipse plug-in model that is used in both client (Workbench) and server (Corona) environments.
- **Collaboration Framework:** Eclipse defines constructs, such as Projects and Resources that can be shared across plug-ins within a single Workbench instance. Corona will extend this model to a distributed environment by allowing these constructs to be shared across multiple Workbench instances. This will enable workgroup collaboration on those particular Resources.
- **SOA Framework:** Corona introduces a web services model that allows Eclipse plug-in extension points to be mapped to web service endpoints, thus allowing non Eclipse-based applications to collaborate with Eclipse-based applications.
- **Semantic Framework:** The semantic framework ties everything together. It provides the persistent memory that describes Resource capabilities as well as relationships between Resources. The semantic framework provides growth opportunities. The knowledge base can be extended to include additional relationships defined by plug-in providers.

providing the communication and collaboration tools used by the special team to share information critical in resolving the problem:

- **Production Monitoring** technology is running on the production server. An exception takes place and the monitoring agent captures key information about the cause of the problem, such as heap information and values of variables.
- The **Production Monitoring Adaptor**, running in the Corona framework, receives the key information via Web Services. The information is converted and published as a **Collaboration Event** within Corona.
- Corona utilizes the OSGi EventAdmin service, implemented by Equinox, to distribute the Collaboration Event to other services that have subscribed to the particular event.
- The **Knowledge Manager** receives a copy of the Collaboration Event that contains the key information. It updates its **Knowledge Base** with the key information and associates it to relevant projects and resources.
- The **Event Router** also receives a copy of the Collaboration Event. It will route the event to all Eclipse Workbench instances that are participating in the workgroup. Developers, team leaders, and managers who are members of the workgroup receive copies of the Collaboration Event. The project manager receives notification of the issue and is able to assign a priority and resource to the issue.
- The developer logs into her **Eclipse IDE** and sees the new issue with the high priority and is able to open the information in her environment and look at the details of the information

captured. If more information is required the developer can send a request to operations to capture additional information, or resolve the problem with the existing information. See [Figure 4](#).

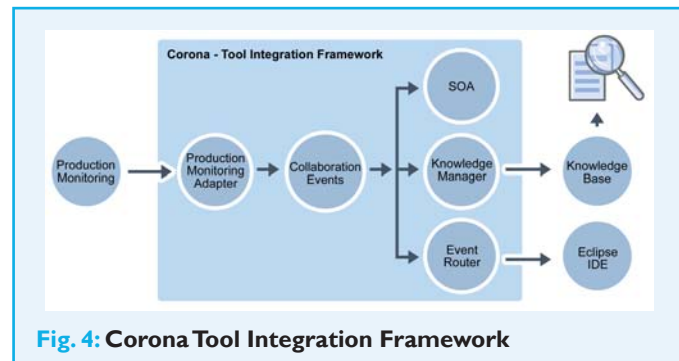


Fig. 4: Corona Tool Integration Framework

The Key Benefits Of Corona

Corona is an open, standards-based extensible collaboration framework. The Eclipse community owns Corona and Compuware is determined to ensure it meets the needs of the market.

Where Eclipse provides interoperability on the Workbench, Corona will extend interoperability into collaboration across Eclipse Workbenches and even across non-Eclipse Workbenches:

- **Vendor benefits:** Corona will allow tools from different vendors to become part of an integrated application life cycle

Feature Enabling Integration and Interoperability for Eclipse-based Development

solution. Every Corona compliant tool will become a potential building block for a life cycle solution. Tools cannot be ignored or excluded anymore as result of not being part of a proprietary ALM solution. By creating Corona plug-ins for their tools, vendors can quickly become part of a bigger solution.

- **End-user benefits:** Corona will allow customers to create their custom application life cycle solution comprised of the tools of their choice. This allows them to reuse the investments they made in tools, methodologies, best practices, and skills as much as possible. It prevents them from having to switch to a single vendor proprietary solution. In addition, Corona will accelerate the speed of delivery for complex, business critical applications, thus improving IT's efficiency and increasing the return of investment (RoI) for the business.

Edwin Schumacher



Based in the worldwide development centre in Amsterdam, the Netherlands, Edwin Schumacher is responsible for the technical direction of Compuware's application development and integration products, including Compuware OptimaJ, an enterprise application development and integration environment that helps organizations adopting J2EE standards increase both developer productivity and Java application quality.



DeveloperSutra



.NET

DATABASE

JAVA

ECLIPSE

PHP

OPEN SOURCE

SECURITY

SOA

WEB TECHNOLOGIES

www.developersutra.com

Information Treatise For The Developer Community

Advertisers

08 JAX Innovation Award
<http://www.jax-award.com>

19 JAX Magazine
<http://www.jaxmag.com>

28 SDA Asia
<http://www.sda-asia.com>

34 Eclipse Magazine
<http://www.eclipsemag.net>

37 SDA India
<http://www.sda-india.com>

48 Developer Sutra
<http://www.developersutra.com>

50 Software &Support Media
www.softwaresupportmedia.com

Imprint

Editor-in-Chief	Indu Britto
Associate Editor	Alexander Neumann
Managing Editor	Dilip Thomas
Authors	Andrey Nechypurenko, Christopher Deckers, Dieter Krachtus, Douglas C. Schmidt, Edwin Schumacher, Egon Wuchner, Frank Schroeder, Jason Weathersby, Jules White, Rajkumar C Madhuram
Layout	S. Ganesh, Gwendolyn Vaz

How to Contact Us

We invite you to submit letters, articles, announcements, and press releases to Eclipse Magazine.

E-Mail: editors@eclipsemag.net
Address: Software & Support Verlag GmbH
Geleitsstraße 14
60599 Frankfurt am Main
Germany
Tel: +49 (0) 69 63 00 89 0
Fax: +49 (0) 69 63 00 89 89
Web Site: www.eclipsemag.net

Advertising

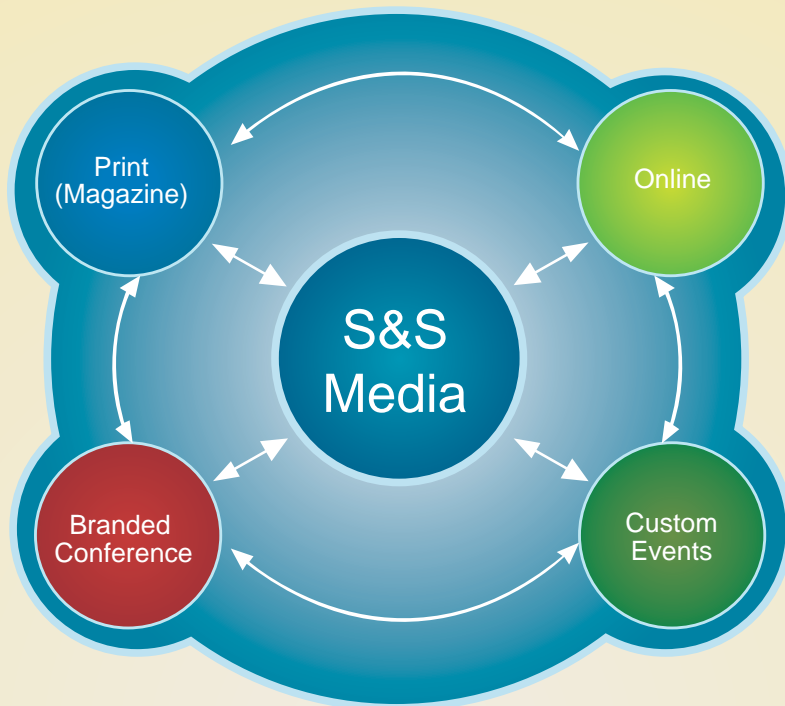
E-Mail: advertise@eclipsemag.net
noutmani@entwickler.com
Tel: +49 (0) 69 63 00 89 0
Fax: +49 (0) 69 63 00 89 89

© Copyright 2007. Software & Support Media

All rights reserved. No part of this publication may be produced in any form without prior consent of the copyright holder. While all reasonable attempts are made to ensure accuracy, Software & Support Media disclaims any liability whatsoever for any use of code or other information therein.

All trademarks and brands used are usually registered trademarks of companies and organisations.

WHERE DO YOU WANT TO MEET YOUR CUSTOMER?



FOUR SERVICES. ONE SOURCE.

S&S Media

CHAMPIONING CROSS MEDIA COMMUNICATIONS

Software & Support Verlag GmbH

Geleitsstraße 14

60599 Frankfurt am Main, Germany

Phone: +49 (0) 69 63 00 89 0 Fax: +49 (0) 69 63 00 89 89

E-mail: sda-asia@software-support.biz

WebSite: www.software-support.biz

S&S Media

#15/6, I FLOOR, PRIMROSE ROAD,

BANGALORE - 560 025

Off: +91 80 41124392/3 Fax: +91 80 41124391

E-mail: editors@sda-india.com

WebSite: www.softwaresupportmedia.in

SINGAPORE S&S MEDIA PTE LTD

133 NEW BRIDGE ROAD #08-10 CHINATOWN POINT

SINGAPORE 059413

TEL: +65 6435 0260 (MAIN LINE) FAX: +65 6887 3842

E-MAIL: advertise@sda-asia.com

WebSite: www.softwaresupportmedia.sg

S&S Media

MENARA KADIN INDONESIA,

JAKARTA 12950, INDONESIA

Phone: +62 21 5263083 Fax: +62 21 5299 4599

E-mail: contactus@media-ti.co.id

WebSite: www.media-ti.co.id

